Swarm-Net: Firmware Attestation in IoT Swarms using Graph Neural Networks and Volatile Memory

Varun Kohli†, Bhavya Kohli†, Muhammad Naveed Aman, *Senior Member, IEEE*, Biplab Sikdar, *Senior Member, IEEE*

***Abstract*—Amidst the large-scale deployment of IoT networks worldwide, studies have highlighted critical security concerns many of which stem from firmware-related issues. IoT swarms have become more prevalent in industries, smart homes, and agricultural applications and malicious activity on one node can propagate to other network sections. While several Remote Attestation (RA) techniques have been proposed in the literature, they are limited by their latency, availability, complexity, hardware assumptions, and uncertain access to firmware copies under Intellectual Property (IP) rights. To address these problems, we present *Swarm-Net*, a novel swarm attestation technique that uses Graph Neural Networks (GNNs) to exploit the inherent, interconnected, graph-like structure of IoT networks and the runtime information stored in the Static Random Access Memory (SRAM). We also present the first datasets on SRAM-based swarm attestation encompassing different types of firmware and edge relationships. In addition, a secure swarm attestation protocol is proposed to ensure authentication, availability, and attestation. *Swarm-Net* is computationally lightweight and does not require a copy of the firmware. It achieves a 99.96% attestation rate on authentic firmware, 100% detection rate on anomalous firmware, and 99% detection rate on propagated anomalies, at a communication overhead and inference latency of ~ 1 second and $\sim 10^{-5}$ seconds (on a laptop CPU), respectively. In addition to the collected datasets, *Swarm-Net*'s effectiveness is evaluated on simulated trace replay, random trace perturbation, and dropped attestation responses, showing robustness against such threats. Lastly, we compare *Swarm-Net* with past works and present a security analysis.**

***Index Terms*—Internet of Things (IoT), Remote Attestation (RA), Swarm Attestation, Graph Neural Networks (GNN), Anomaly Detection, Static Random Access Memory (SRAM)**

I. Introduction

The Internet of Things (IoT) has emerged as a leading force behind smart city initiatives for agriculture, healthcare, homes, industry, transportation security, and supply chains, owing to advances in 5G, artificial intelligence, and edge computing technologies [1]. Despite billions of dollars worth of investment worldwide [2], its constituent devices, which are mostly low-end embedded systems with limited computational capabilities, are often the target of various security threats [3].

A recent study highlighted that 95% of security vulnerabilities in IoT networks stem from firmware-related problems [4]. Thus, evaluating firmware integrity is an essential part of ensuring security and trust in IoT networks. Remote Attestation (RA) has emerged as an important field of research for this purpose, and various single-node software [5], hardware [6], or hybrid [7] RA techniques have been proposed. However, their applicability to larger networks is often naive, and therefore, efficient swarm attestation approaches have also been proposed [8]. Among available attestation techniques, several software-based methods rely on computationally intensive checksums over the IoT device's flash memory and offer low availability. They also require a copy of the firmware, which may not be possible under the manufacturers' Intellectual Property (IP) rights [9–11]. Further, hardware and hybrid RA techniques assume the availability of Trusted Platform Modules (TPM), Memory Protection Units (MPU), and Trusted Execution Environments (TEE), which do not apply to low-end devices [12–14]. Lastly, studies based on network-flow information have limited detection capabilities against malicious firmware with normal network activity [15].

A recent paper [16] made minimal hardware assumptions on IoT devices and used Static Random Access Memory (SRAM) as a feature for machine learning-based attestation. However, their method targeted single-node RA, used the entire SRAM, and was limited to a few anomalous firmware classes in a classification-based approach. We show that the SRAM data section (a smaller part of the SRAM) is useful for detecting nodes infected by malicious firmware and propagating anomalies when combined with Graph Neural Networks (GNNs) for anomaly detection. In addition, the SRAM is significantly smaller than flash memory and, thus, easier to traverse. It captures runtime information, can indicate roving malware, and eliminates the need for firmware copies during attestation. Furthermore,

Contributions: To the best of our knowledge, this paper is the first to use SRAM as a feature for swarm attestation and presents the first datasets on SRAM-based swarm attestation. The contributions of this study are listed below.

1) A novel GNN-based lightweight method for firmware attestation in swarms. The proposed method uses the SRAM's data section and simple GNN designs.

This research was supported in part by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Future Communications Research Development Programme, under grant FCP-NUSRG-2022- 019. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Infocomm Media Development Authority.

V. Kohli and B. Sikdar are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117417. (e-mail: varun.kohli@u.nus.edu, bsikdar@nus.edu.sg).

B. Kohli is with the Center for Machine Intelligence and Data Science, Indian Institute of Technology Bombay, Mumbai, India 400076. (e-mail: bhavyakohli@iitb.ac.in)

M. N. Aman is with the School of Computing, University of Nebraska-Lincoln, Lincoln, NE 68588, USA. (e-mail: naveed.aman@unl.edu).

† marked authors have made equal contribution to this paper.

TABLE I: Qualitative comparison of related works on single-node and swarm attestation.

Evaluation Criteria	[9–11]	[16]	[12, 17, 18]	[13, 19, 20]	HAtt [21]	[22–28]	WISE [29]	FeSA [30]	[15]	RAGE [31]	[14]	Swarm-Net
Target	Single	Single	Single	Single	Single	Swarm	Swarm	Swarm	Swarm	Single	Swarm	Swarm
RA methodology*	S/W	S/W	H/W	H/b	H/b	H/b	H/b	S/W	S/W	H/b	S/W	S/W
Approach*	Crypto	ML	Crypto	Crypto	Crypto	Crypto	Crytpo + ML	FL	GNN	GNN	ML	GNN
Feature used	Flash	SRAM	Flash	Flash	Flash	Flash	Flash	Property	Network	Execution trace	Property	SRAM
Low availability	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
High latency	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Interrupts disabled	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Specific hardware	✗	✗	✓	✓	✗	✓	✓	✗	✗	✓	✗	✗
Homogenous devices	✗	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗
Firmware needed	✓	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
Limited attacks	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗

* Notations - S/W: Software, H/W: Hardware, H/b: Hybrid, Crypto: Cryptography, ML: Machine Learning, FL: Federated Learning, GNN: Graph Neural Network

2) A secure swarm attestation protocol and the corresponding security analysis.
3) Datasets on SRAM-based swarm attestation for (physically deployed) four four-node and six-node swarm configurations, encompassing various complex node-level and inter-node behaviors.
4) Thorough experimental results of different GNN architectures using the collected datasets and additional simulated scenarios.
5) A comparison with related works on remote and swarm attestation.

The codes for firmware, dataset collection, and attestation are available on GitHub[1]. The remainder of this paper is organized as follows: Section II discusses related works on attestation and presents a qualitative comparison of the proposed method with related works. Section III discusses background concepts on SRAM and GNN, followed by Section IV, which presents the network and threat models considered in this study. Section V introduces, *Swarm-Net*, the proposed swarm attestation method, followed by a detailed discussion on the dataset, devices, hyperparameters, and evaluation metrics in Section VI. Section VII shows the experimental results, and Section VIII analyses the security of the proposed method. The paper is concluded in Section IX.

II. Related Works

Several techniques for single-node RA and swarm attestation have been proposed to solve firmware-related problems in IoT. We structure our discussion into three parts: (a) on specific software (S/W), hardware (H/W), and hybrid (H/b) RA techniques for single prover scenarios, followed by (b) swarm attestation approaches, and lastly, (c) a brief overview of *Swarm-Net*. Table I presents a qualitative comparison of *Swarm-Net* with related works.

A. RA methodologies

Traditional software-based RA techniques in single-prover scenarios typically require an IoT microcontroller to evaluate a checksum on its program memory, which the verifier validates using a copy of the hash digest [9–11]. SWATT [9], for instance, performs over fifty thousand cumulative, uninterrupted hash iterations using pseudo-random memory traversal to attest a device. Attacks are detected by calculating time deviations in normal and attack scenarios. However, such an approach is complex, has high latency ($\sim$ minutes), low IoT device availability, uses precise time measurements, and requires a copy of the firmware with the verifier. SCUBA [10] and SAKE [11] share similar problems. A few lightweight software attestation approaches use a combination of the microcontrollers' flash and RAM for firmware attestation using checksum verification but do not consider roving malware [32–34]. The authors of [16] use the SRAM as a feature for a Multi-layer Perceptron (MLP) classifier to attest among known malware and authentic nodes at high accuracy and availability; however, their method has limited scalability to a larger number of devices and firmware variants due to its classification-based approach. Notable studies on hardware-based RA include [12, 17, 18], which assume the availability of a TMP or software enclaves. This assumption does not apply to medium- and low-end IoT devices. To find a middle ground between hardware assumptions and cost-effectiveness, various hybrid approaches based on hardware/software codesign have been proposed, among which some notable studies include SMART [19], TrustLite [20], TyTan [13], ATT-Auth [35], HAtt [21] and [36]. [13, 19, 20, 35] however, do not provide security against roving malware. Although the RAM-based approach presented in [36] has low latency, it requires a memory protection unit for privileged/unprivileged software execution. HAtt [21] is another lightweight mechanism that offers a high availability of IoT nodes. It uses Physically Unclonable Functions (PUF) to protect secret information on IoT nodes from physical attacks, although requiring a copy of the microcontroller's firmware. The authors of a recent study propose RAGE [31], a hybrid, Variational Graph Autoencoder (VGAE)-based control-flow attestation method that attests IoT devices using a single execution trace collected from each device using a TEE. They achieve an average performance of 91% and 98% for return-oriented programming and data-oriented programming attacks, respectively, with high availability. However, the TEE assumption does not apply to low-end devices.

B. Swarm attestation

The feasibility and efficacy of RA techniques designed for single-node scenarios do not always relate to larger networks - uniform assumptions may not apply to networks of heterogeneous devices with different communication and hardware capabilities. In addition, the attestation feature used is of great

[1]https://github.com/VarunKohli18/swarm-net

importance; for instance, leaving aside their latency, software-based approaches that use program memory checksums cannot capture relationships between nodes. These issues highlight the importance of efficient swarm attestation using suitable features; several (mostly hybrid and program memory checksum) methods have been proposed.

Notable cryptographic methods include [22–28]. The authors of [22] propose a distributed, Merkle Hash Tree (MHT) attestation process for in-vehicle controller area networks in which. However, the nodes have high complexity and memory requirements. Furthermore, points of failure can lead to various unattested nodes. SEDA [23] is a hybrid swarm RA technique built upon extends SMART [19] and TrustLite [20]. The verifier selects an arbitrary initiator node, creating a spanning tree following it. Each node calculates a hash over its memory, and the initiator collects the accumulated attestation report and shares it with the verifier for cross-checking. Despite its viability, a few limitations include its architectural impact, attestation timeout selection, and initiator node selection as highlighted by [24]. The latter study proposes asynchronous and synchronous protocols called LISAα and LISAs, respectively, with minimal changes over SMART+ [24]. SAFE^d [25] eliminates the need for a verifier by spreading security proofs among devices in a swarm, which they can use to validate each other. Another study, HEALED [28], can detect malicious firmware using MHT on the nodes' flash memory and disinfect compromised nodes. The five discussed studies, however, do not account for roving malware.

Some Machine Learning (ML)-based approaches have also been proposed. WISE [29] is an intelligent attestation method that addresses the heterogeneity of IoT devices and roving malware using a multi-clustering technique and variable attestation windows. It achieves an average detection rate of 62%, with an evaluation latency of about 3.5 seconds. FeSA [30] is a distributed swarm attestation technique that uses Federated Learning (FL) and dynamic attestation periods based on the properties (device state, energy, and traffic) and security requirements for different devices, achieving an average of 87% accuracy across various scenarios and low inference latency ($\sim$ one second). However, federated learning is typically associated with lower accuracies. A GNN-based distributed anomaly detection approach, is presented in [15]. The proposed graph MLP achieves an average 97.7% accuracy for various network attacks using five or more seconds of network flow data. However, the model has limited applicability to malicious firmware, such as those with normal network behavior (which is a limitation of all network flow-based methods). Lastly, a TEE assumption is made in [14], which uses machine learning for property-based attestation.

C. Swarm-Net

Based on the above discussion, most studies have limitations in one or more properties shown in Table I, aside from some latency and performance differences, which we highlight in Section VII. While there exists a study on single-node attestation using SRAM [16], there are no available studies on SRAM-based attestation of swarms. To fill these gaps, we propose a novel GNN- and software-based attestation approach for reliable swarm attestation with high availability. *Swarm-Net* makes minimal assumptions on IoT devices and verifies their state using a single SRAM trace during the attestation phase. We justify the use of SRAM over flash memory and network flow features for the following reasons:

1) SRAM is a readily available component on all medium- and low-end IoT microcontrollers.
2) It is significantly smaller than flash memory by several orders of magnitude and is, therefore, faster to traverse.
3) It captures runtime information, and its usage does not depend on the location of malware in the memory. It can indicate any firmware-related anomaly on the microcontroller, including roving malware.
4) Since received messages are typically stored in the SRAM for usage, propagated anomalies are detectable.
5) Lastly, there is no IP violation since a copy of the firmware is not required.

Since the proposed method does not require the IoT device to compute uninterrupted checksums over the memory, it offers high availability on the IoT device. Furthermore, the interconnected nature of IoT networks is exploited using GNNs, enabling *Swarm-Net* to (both) detect node-level and propagated anomalies.

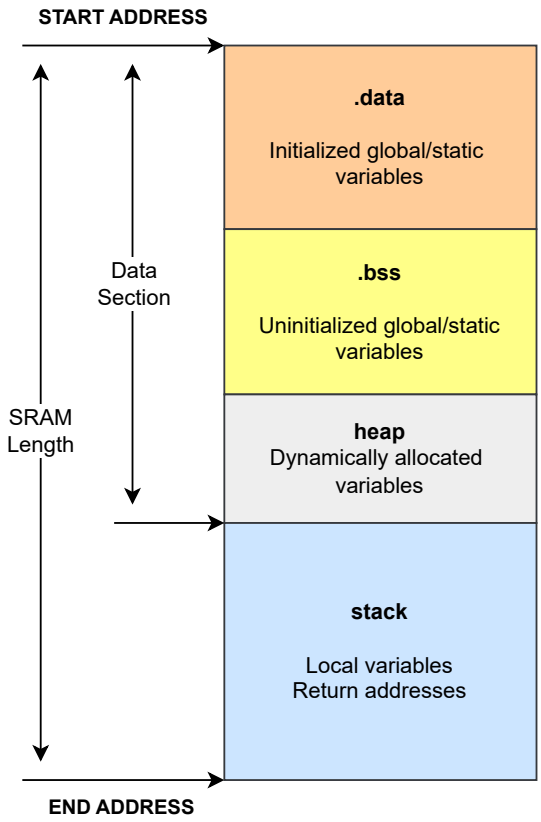


Fig. 1: General organization of a microcontroller's SRAM.

III. Background

This section provides a basic background on SRAM and GNN; Table II compiles a list of relevant notations for the

TABLE II: List of relevant notations.

Notation	Description
MSE	Mean Square Error
CS	Cosine Similarity
GCN	Graph Convolution Network
GAT	Graph Attention Network
GT	Graph Transformer
ID_V	Verifier ID
ID_G	Gateway ID
ID_S	Swarm ID
N_j	j^{th} node in a swarm
T_j	Complete SRAM trace
T'_j	SRAM data section trace
b_i	Hex value at i^{th} byte
d	Data section length
G_θ	GNN with parameters θ
$\mathcal{N}_v$	Number of one-hop neighbors
e_{ij}	Attention coefficients
α_{ij}	Importance values
W	Weight matrix
SR	Swarm Response
X	Trainset
x	Graph sample
$\hat{x}$	Reconstructed graph
$\tilde{x}$	Graph sample with uniform noise
ϵ	Uniform noise
k	Noise factor
DT	Decision Threshold
sf	Scaling factor
T_{def}	Default trace set
L	Maximum padding length
f	Predicted decision flags
C	Nonce
AN_j	Anomaly at N_j
S_i	Simulated scenario i
AR	Attestation Rate
DR	Detection Rate
T_o	Overhead/latency

readers' reference.

A. Static Random Access Memory

SRAM is a type of volatile memory that stores an embedded device's runtime data and is refreshed on every powerup. It is several orders of magnitude smaller than flash (thus, more practical), and its contents are refreshed on every power-up. The SRAM stores useful runtime information about the firmware installed on a microcontroller and is divided into four segments [37] as shown in Figure 1. The .data and .bss sections store initialized and uninitialized global or static variables, respectively. The heap stores dynamically allocated variables, and the stack stores local variables and return addresses. The first three sections together may be called the data section, which has a varying size depending on the memory usage of the program loaded on the microcontroller. As we show in this paper, the data section is a useful feature for verifying firmware integrity. The stack, on the other hand, has been used for device authentication by a recent study [38].

The data section has the same behavior for the same firmware on different devices (consistency) [38] and different behavior for different firmware versions on the same device (distinguishability) [16]. Making changes to the firmware leads to changes in the SRAM during runtime, and thus, malicious code is likely to have abnormal SRAM dumps. Furthermore, in a swarm setting, data generated at one node and sent to another creates a relationship between the otherwise independent SRAM contents of the two nodes as variables with the same values are created on both devices. This can help detect the downstream effects of firmware and network anomalies. We leverage these properties to perform software-based swarm attestation of microcontroller firmware.

SRAM notations: A complete SRAM trace (t_j) generated by an IoT device N_j is represented as a sequence of l bytes:

$$T_j = [b_0, b_1, ..., b_{l-1}]_j \ , \tag{1}$$

where b_i is the hexadecimal value stored at the i^{th} byte of the SRAM, and l is the length of the node's SRAM. However, each node in the swarm must send only the SRAM data section:

$$T'_j = [b_0, b_1, ..., b_{d-1}]_j \ , \tag{2}$$

where d is the size of the data section corresponding to the firmware loaded on N_j. Note that d ¡¡ l, typically.

B. Graph Neural Networks

Graph notations: We denote a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set consisting of nodes $\{v_1, v_2, ..., v_n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents the set of edges in $\mathcal{G}$. For a given node $v \in \mathcal{V}$, $\mathcal{N}_v$ denotes the set of one-hop neighbors of v, i.e., all nodes $w \in \mathcal{N}_v$ are connected to v via one edge. We also denote $\mathcal{A} \in \{0,1\}^{n \times n}$ as the adjacency matrix and $\mathcal{X} \in \mathbb{R}^{n \times d}$ as the node features of the graph. In this study, we assume a given graph is simple and unweighted.

Network architectures: GNNs typically follow the message-passing propagation framework, which first aggregates the features of a node v with neighboring nodes $u \in \mathcal{N}_v$ using an aggregation function ϕ, then updates them using a learnable function f_θ (usually an MLP) [39]. This process is repeated several times to obtain node representations that can be used for downstream tasks.

GNNs are suitable for analysing the SRAM behavior of IoT swarms which typically have a graph-like structure. Edges between IoT devices can have complex behaviors that need to be modeled using suitable learnable aggregation methods for a more robust understanding of how the network shares and uses information. In this paper, we consider three GNN architectures from Pytorch Geometric (PyG) [2] for attestation: Graph Convolution Networks (*GCN*) [40], Graph Attention Networks (*GAT*) [41] and Graph Transformers (*GT*) [42].

Graph Convolution Network: Given node features $H^{(l)}$ at layer l (sufficiently padded to account for bias), feature propagation in the case of the PyG implementation *GCNConv* is computed as follows:

$$H^{(l+1)} = \sigma(\tilde{\mathcal{D}}^{-1/2}\tilde{\mathcal{A}}\tilde{\mathcal{D}}^{-1/2}H^{(l)}W^{(l)}) \ , \tag{3}$$

[2]https://pytorch-geometric.readthedocs.io/en/stable/

where $\tilde{\mathcal{A}} = \mathcal{A}+\mathcal{I}$ (self-loops added), $\tilde{\mathcal{D}_{ii}} = \sum \tilde{\mathcal{A}}_i$, $W^{(l)}$ is the weight matrix for layer l, and $\sigma(\cdot)$ is an activation function.

Preliminary intuition: GCN assigns equal importance to all neighbors [41], and despite its inherent simplicity which makes it worth testing in this use case, it may not be ideal for heterogeneous inter-node relationships present in IoT swarms, which is supported by our results in Section VII. We therefore also look at other GNN architectures, such as GAT and GT, that provide better parameterization of graph edges.

Graph Attention Network: GAT computes attention coefficients e_{ij} as per the following:

$$e_{ij} = a(Wh_i, Wh_j) \ . \tag{4}$$

The attention mechanism $a(\cdot)$ uses a weight matrix W, and the current node features h_i and h_j to compute the *importance* of node j to node i. These important values α_{ij} are normalized across the neighborhood of node i. The PyG implementation *GATConv* computes α_{ij} using the following:

$$\alpha_{ij} = \text{softmax}_i(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})} \ . \tag{5}$$

Graph Transformer: Transformer networks have emerged as the state-of-the-art for various machine learning tasks due to their ability to handle multimodal data and their improvement over Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) architectures in learning long-term dependencies in sequential data [43]. We use the Unified Message Passing model (UniMP), which extends vanilla multi-headed attention [44] to graphs. The PyG implementation of GT, *TransformerConv*, uses 4 weight matrices $W_1^{(l)}, W_2^{(l)}, W_3^{(l)}, W_4^{(l)}$ per attention head at layer l to compute a weighted aggregation of features of neighboring nodes. Additionally, a fifth and a sixth weight matrix ($W_5^{(l)}$ and $W_6^{(l)}$) can be introduced if gated residual connections between layers or edge features need to be considered. Attention coefficient coefficients α_{ij}'s are computed (softmax over all nodes $j \in \mathcal{N}_i$) using

$$\alpha_{ij} = \text{softmax}_i\left(\frac{q^T k_j}{\sqrt{C}}\right) \ , \tag{6}$$

where $q = W_1^{(l)}h_i^{(l)}$, $k_j = W_2^{(l)}h_j^{(l)}, v_j = W_3^{(l)}h_j^{(l)}$, and feature propagation follows

$$h_i^{(l+1)} = W_4^{(l)}h_i^{(l)} + m_j \ , \tag{7}$$

where $m_j = \sum_{j\in\mathcal{N}_i} \alpha_{ij}v_j$. Here C is the number of output channels for a given *TransformerConv* layer. For more than one head, unique q and k_j values are computed for each head, and the final m_j is obtained by either concatenating or averaging all $m_j^{(c)}$'s.

Preliminary intuition: In contrast to GAT, which uses a single shared linear transformation on every node parameterized by a single weight matrix W, feature propagation in GT may enable learning more complex inter-node relationships due to a higher degree of parameterization using $W_{1-4}^{(l)}$ [44]. This intuition is supported by our results presented in Section VII.

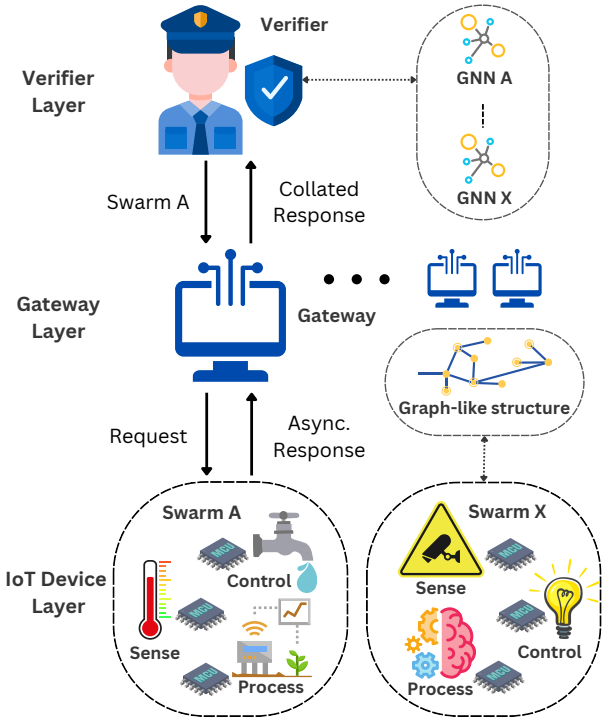


Fig. 2: Layered network model.

IV. Network and Threat Model

This section presents the network model and assumptions about an adversary.

A. Network Model

Figure 2 shows the network model considered in this study. The network consists of three overall layers:

1. **Verifier Layer:** This layer comprises the verifier, a trusted remote device with computational power to run the proposed attestation method. The verifier sends attestation requests for a specific swarm to the gateway devices under its jurisdiction. It analyzes the collated responses from the gateways using the corresponding GNN. We assume that the communication between the verifier and the gateways is secure and that the verifier has prior knowledge of the functionality of the swarm and its constituent devices [30].
2. **Gateway Layer:** This layer consists of one or more trusted gateway devices that receive attestation requests from the verifier for the swarms under their jurisdiction, send an attestation request to the swarm specified by the verifier, collect the asynchronous responses received from a swarm's devices, and send the collated response set to the verifier.
3. **IoT Device Layer:** This layer comprises swarms of vulnerable IoT devices, each comprising a microcontroller as the hardware platform. The tasks of the IoT devices may be divided into three overall categories (or their combinations): sense, process, and control [24]. In the context of attestation, IoT devices are the provers that respond to attestation requests from the gateway with

Algorithm 1: Training algorithm run by ID_V.

$t \leftarrow time()$
$request(ID_G, ID_S, samples = m)$
while $time() - t \geq timeout$ **do**
  if $receive(sender = ID_G)$ **then**
    $SR \leftarrow receive(sender = ID_G)$
    $break()$
  else
    $return(-1)$ `// No reply from` ID_G
$G_\theta \leftarrow initializeGNN()$
$X \leftarrow pad(int(SR)/255, L)$
$\tilde{x} = x + k \cdot \epsilon, \forall x \in X$
$e = 0$
while $e \leq epochs$ **do**
  $G_\theta \leftarrow backprop(\theta, x, \tilde{x}), \forall x \in X$
  $e = e + 1$
$DT_j \leftarrow sf \cdot \min_{i \in [0 \ldots m-1]} CS(X_j, \hat{X}_j), \forall N_j \in ID_S$
$T_{def,j} = \frac{\sum X_j}{m}, \forall N_j \in ID_S$
$DT \leftarrow [DT_0, ..., DT_n - 1]$
$T_{def} \leftarrow [T_{def,0}, ..., T_{def,n-1}]$
$saveParams(\{G_\theta, DT, T_{def}, L\}_S)$

Algorithm 2: Attestation algorithm run by ID_V.

$t \leftarrow time()$
$request(ID_G, ID_S, samples = 1)$
while $time() - t \geq timeout$ **do**
  if $receive(sender = ID_G)$ **then**
    $SR \leftarrow receive(sender = ID_G)$
    $break()$
  else
    $return(-1)$ `// No reply from` ID_G
$\{G_\theta, DT, T_{def}, L\}_S \leftarrow loadParams(ID_S)$
$x \leftarrow SR == 0 \ ? \ T_{def} : SR$
$x \leftarrow padding(int(x)/255, L)$
$\hat{x} \leftarrow G_\theta(x)$
$f \leftarrow CS(x, \hat{x}) > DT \ ? \ 0 : 1$
$return(f)$

their SRAM dumps. Swarms have an inherent, graph-like structure due to the necessary communication between sense-process-control tasks. Furthermore, we assume that the direction and type of information flow in the swarm are known to the verifier, given its knowledge of the nodes' normal functionality at the time of deployment.

B. Threat Model

The following assumptions are made about the adversary.

1. The adversary may send malicious firmware updates to one or more remote devices. In doing so, the adversary may partially or completely change an IoT device's functionality.
2. The adversary can eavesdrop on the communication within the swarm and launch man-in-the-middle attacks.
3. The adversary can drop messages shared between (a) two IoT devices to create out-of-sync network states and (b) an IoT device and the gateway to cause desynchronization of the attestation round.
4. The adversary can launch a denial of service on the swarm by impersonating the gateway and frequent attestation requests. It may achieve this by replaying previously authorized attestation requests sent to the swarm.
5. The adversary can replay old attestation responses to impersonate a prover.
6. The adversary cannot access the secret key or the attestation round nonce shared between the IoT microcontrollers and the trusted gateway [45].

Swarm-Net attempts to solve these threats and is discussed in Section V.

V. Proposed Technique: Swarm-Net

This section discusses the training phase, the attestation algorithm, and the overall attestation protocol.

A. Training Phase

After deploying a swarm ID_S, the verifier ID_V initiates the training phase depicted in Algorithm 1. It prompts the gateway ID_G to collect a sufficient number ($m \gtrsim 500$ based on experiments) of collated samples from each IoT device N_j under normal operation. ID_G collects the necessary m-sample swarm response SR and sends it to ID_V. We assume this collection happens under monitored conditions with no malicious activity to ensure a clean training dataset. On the verifier's end, each T' in SR is converted to its integer equivalent, scaled by a factor of 255 (to bring it in a [0,1] range), and padded to an appropriately selected maximum length L for uniformity during evaluation. A training set of graphs $X \in \mathbb{R}^{m \times n \times L} \subset \mathbb{X}$ is thus created, where $\mathbb{X}$ is the complete distribution of graphs associated with normal firmware in the swarm. The learnable parameters θ are then optimized using back-propagation to approximately reconstruct the training distribution $\mathbb{X}$. To do so all $x \in X$ are perturbed with scaled Uniform noise $\epsilon \sim \mathcal{U}(0, 1)$ such that $\tilde{x} = x + k \cdot \epsilon$ where $\epsilon, x \in \mathbb{R}^{n \times L}$ and the GNN is trained to denoise $\tilde{x}$ such that $MSE(x, \hat{x})$ is minimized. Here, k is the noise factor, $\hat{x} = G_\theta(\tilde{x})$ and $MSE(x, \hat{x})$ is computed as follows:

$$MSE(x, \hat{x}) = \frac{||x - \hat{x}||_2^2}{nL}, \tag{8}$$

where $|| \cdot ||_2$ is the L_2 norm. Adding ϵ is necessary to force G_θ to learn the output distribution $\mathbb{X}$ regardless of the input, thereby improving the model's performance on anomalous samples. Upon completion of model training, ID_V computes the detection threshold DT_j of each device N_j. While most anomaly detection studies use MSE-based thresholds, we observe that Cosine Similarity (CS) enables a more intuitive and generalized threshold selection approach. Each DT_j is computed as follows:

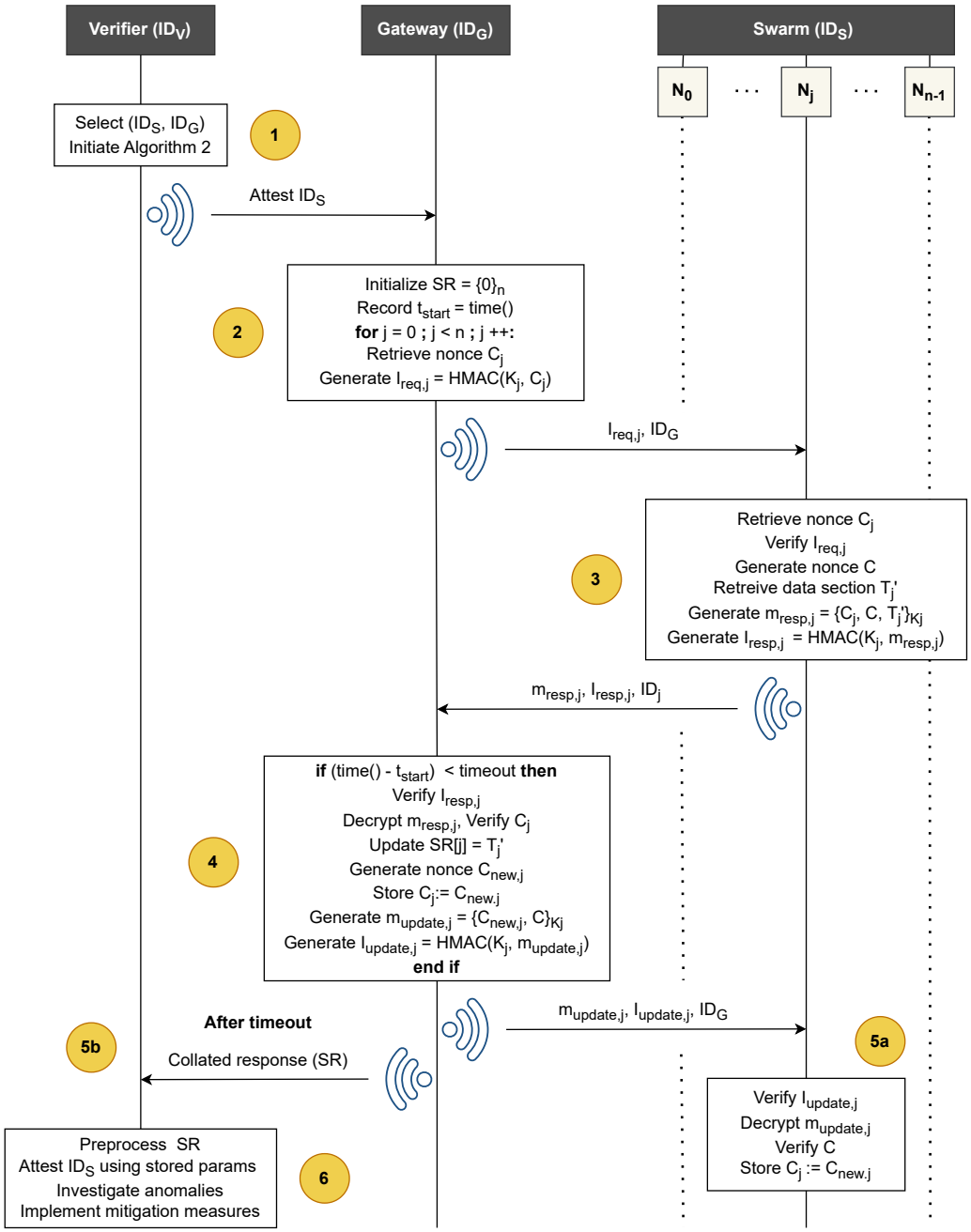


Fig. 3: The *Swarm-Net* attestation protocol.

$$DT_j = sf \cdot \min_{i \in [0 \ldots m-1]} CS(X_j, \hat{X}_j) \ , \tag{9}$$

where X_j is the preprocessed data of N_j, and sf is the scaling factor of the thresholds, which decides how close the decision boundary should be to the normal CS scores. We use $sf = 0.999$ in our experiments. Further, $\hat{x} = G_\theta(x)$ and CS between two vectors $\bar{u}$ and $\bar{w}$ is evaluated as:

$$CS(\bar{u}, \bar{w}) = \frac{\bar{u} \cdot \bar{w}}{|\bar{u}||\bar{w}|} \ . \tag{10}$$

ID_V compiles each DT_j into a single set $DT \in \mathbb{R}^{n \times 1}$. Further, for the event of dropped attestation responses from nodes in ID_S, ID_V computes a set $T_{def} = [T_{def,0} \ldots T_{def,n-1}] \in \mathbb{R}^{n \times L}$ of default traces where $T_{def,j} \in \mathbb{R}^{1 \times L}$ is computed as:

$$T_{def,j} = \frac{\sum X_j}{m} \ . \tag{11}$$

Here, X_j is the training data of N_j. These average traces are in the event of dropped responses during attestation. This way, ID_V can verify the firmware of other respondents while avoiding a single point of failure. Finally, ID_V stores the parameter set $param = \{G_\theta, DT, T_{def}, L\}_S$ in its memory. In addition, ID_G creates a set of resynchronization nonces L_R, which it distributes to the swarm. L_R is used by ID_G to resync the attestation process in case of desynchronization

attacks by an adversary.

B. Attestation Phase

The attestation phase includes two sub-parts: an algorithm and the overall protocol.

*1) **Attestation algorithm**:* Algorithm 2 presents the attestation procedure run by ID_V to attest ID_S. The verifier requests ID_G for one graph SRAM sample from ID_S over a secure channel and waits for ID_G's collated swarm response SR. Upon receiving SR, it retrieves $param = \{G_\theta, DT, T_{def}, L\}_S$ and preprocesses SR. It replaces the missing responses (if any) using the respective default values of the nodes in the set T_{def}. It then scales the traces by a factor of 255 and pads them with zeros to a common length L. The model G_θ then generates a reconstructed graph $\hat{x} = G_\theta(x)$. ID_G evaluates the similarity scores of each node as $CS(x, \hat{x})$ and compares them with the pre-determined DT. All N_j with CS_j above their respective DT_j are flagged as authorized (0) and, otherwise, anomalous (1). The algorithm returns the decision array f, which the verifier uses to investigate anomalies (if any) and initiate necessary threat mitigation measures. Algorithm 2 is encapsulated within the attestation protocol, which is explained next.

*2) **Attestation protocol**:* The *Swarm-Net* attestation protocol is presented in Figure 3. The devices use symmetric-key encryption and Hash-based Message Authentication Codes (HMAC, SHA-256 [46]). Furthermore, we assume that the gateway securely shares the secret keys with the IoT devices during the initial setup and as needed in subsequent firmware updates [45]. One run of the protocol consists of the six following steps:

1. The verifier ID_V picks a swarm ID_S and its corresponding trusted gateway ID_G. It then initiates Algorithm 2, sends an attestation request to ID_G for ID_S via a secured channel and waits for a response.
2. ID_G initializes a n-size Swarm Response (SR) and begins the timeout timer. It retrieves a stored nonce C_j and the shared secret K_j and generates an HMAC $I_{req,j}$ for each node N_j in the swarm, which it then sends to the nodes.
3. Each N_j verifies the validity of HMAC $I_{req,j}$ using the stored C_j and shared secret key K_j. It generates a nonce C and retrieves its SRAM data section T'_j. It sends an encrypted message $m_{resp,j}$ and the HMAC $I_{resp,j}$ to ID_G.
4. ID_G collects the responses from all nodes and accepts them if they arrive within the timeout. It verifies the accepted HMAC $I_{resp,j}$ using K_j. It decrypts $m_{resp,j}$ and verifies C_j. It then generates a new nonce $C_{new,j}$ for each valid respondent N_j, stores it into C_j, and creates an encrypted message $m_{update,j}$ and HMAC $I_{update,j}$ which it sends to N_j.

5a. Each respondent N_j verifies the received $I_{update,j}$ using K_j. It decrypts $m_{update,j}$, verifies C, and stores $C_{new,j}$ into C_j for the next attestation round.

5b. Upon completion of the timeout, ID_G sends the collated SR to ID_V over a secure channel.

6. ID_V continues Algorithm 2. It preprocesses SR to x and evaluates the reconstructed traces $\hat{x}$ using the stored G_θ. It then evaluates a trust decision f for the swarm using DT. In case of anomalies, the administrator is prompted to investigate and take mitigation measures.

VI. Experimental Setup

This section covers the experimental setup used in this paper, including the devices, programming platforms and libraries, sample IoT swarms, datasets, hyperparameters, and evaluation metrics.

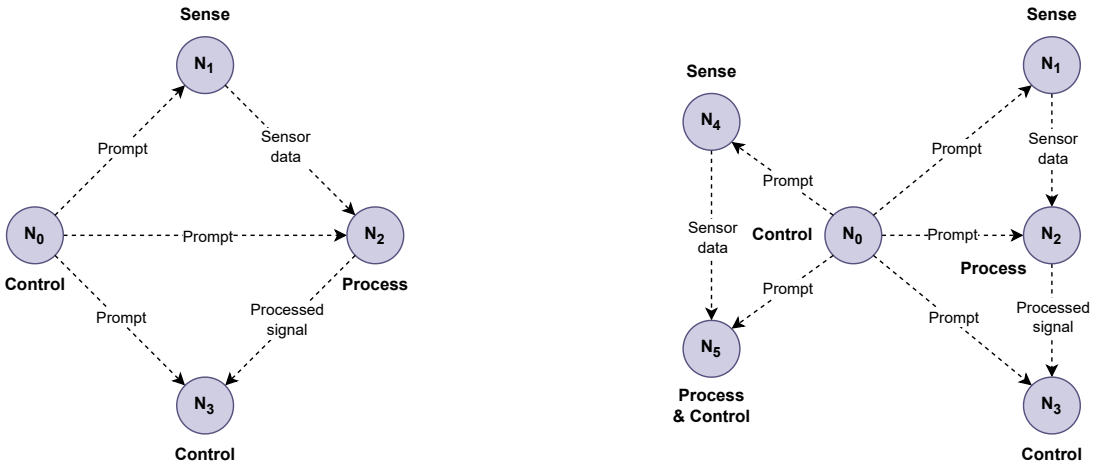
A. Hardware and Software

Figure 4 and 5 show sample swarms and the physical setup used for dataset collection, respectively. All IoT nodes are variants of the Arduino UNO Rev3 and Elegoo UNO Rev3 devices, each with an 8-bit ATmega328P microcontroller and a 2KB SRAM. To understand the efficacy of the IoT device SRAM as a suitable feature for swarm attestation, we aimed to use simple devices and simplified representations of IoT firmware and network behaviors in this paper. Therefore, Arduino UNO Rev3 devices were selected for this study due to their inherent simplicity and flexibility. The gateway device is a Dell Latitude laptop with an Intel i7 processor and 16GB DRAM. Further, we use two verifier devices: GPU experiments were run on a compute server with an AMD EPYC 7742 processor and an Nvidia A100-SXM4 GPU, and CPU experiments were run locally on a laptop with an Intel(R) Core(TM) Ultra 7 155H processor.

The firmware was programmed in Arduino IDE 2.3.2, while the data collection and attestation codes were written in Python 3.8. The main Python libraries used are Numpy 1.26.4, Pandas 2.2.2, Pyserial 3.5, and Pytorch Geometric 2.5.3.

B. IoT Swarms, Datasets, and Simulations

We present our SRAM dataset on IEEE Dataport [47]. The gateway device used a Python script to send attestation requests to the Arduino UNO devices in the physical network (Figure 5) using the Pyserial Python library. The IoT devices then printed their SRAM contents onto their serial monitors for the gateway to read and store. This process was repeated to create a corpus for various normal and abnormal network states by flashing one or more devices with anomalous firmware variants, as explained later in this subsection. The sample networks shown in Figure 4 are simplified representations of real-world IoT swarms in smart city initiatives wherein each device performs one or more of three overarching IoT tasks - sense, process, and control. While practical smart IoT implementations may be more complex, it is fruitful to consider networks such as Figure 4a and 4b as a proof of concept. Furthermore, the gateway device collected unencrypted SRAM samples from the sample swarms to save development and sampling time. However, in a real-world setting, the device and firmware manufacturers must ensure that their IoT devices also have the functionality to ensure confidentiality and integrity of communication.



(a) Four-node network configuration of Swarm-1.

(b) Six-node network configuration of Swarm-2.

Fig. 4: Swarm configurations used to collect the two datasets.

TABLE III: Normal and anomalous firmware in Swarm-1.

Node	Type	Normal firmware functions	d	Anomalous firmware functions	d
N_0	Control	Broadcasts one byte to all nodes	191	Generate three random integers Broadcasts one byte to all nodes	195
N_1	Sense	Generate six floating point numbers in unique ranges Send data to N_2	450	Generate data in an extended range Send data to N_2	438
N_2	Process	Process received data into a six-byte signal Send processed signal to N_3	516	Generate a random six-byte signal Send control signal to N_3	414
N_3	Control	Control six LEDs using the processed signal	406	Control six LEDs at random	386

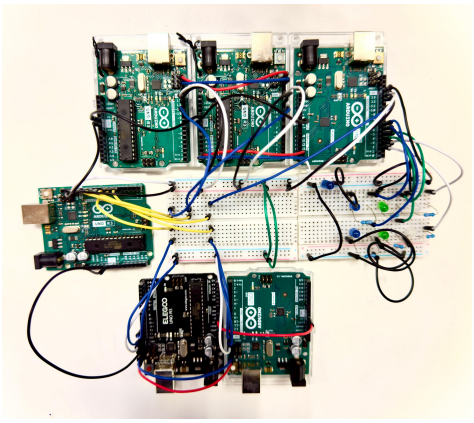


Fig. 5: Physical swarm setup for dataset collection.

TABLE IV: Swarm-1 dataset scenarios.

S.No.	Scenario	Primary Anomaly	Secondary Anomaly	Swarm Label $N_0 - ... - N_3$
1-2	D_{1-2}	-	-	0-0-0-0
3-4	P_{1-2}	-	-	0-0-0-0
5	AN_0	N_0	-	1-0-0-0
6	AN_1	N_1	N_2, N_3	0-1-1-1
7	AN_2	N_2	N_3	0-0-1-1
8	AN_3	N_3	-	0-0-0-1
9	AN_{12}	N_1, N_2	N_3	0-1-1-1
10	AN_{23}	N_2, N_3	-	0-0-1-1
11	AN_{13}	N_1, N_3	N_2	0-1-1-1
12	AN_{123}	N_{1-3}	N_2, N_3	0-1-1-1
13	AN_{0123}	N_{0-3}	N_2, N_3	1-1-1-1

*1) **Swarm-1**:* Swarm-1 is the four-node network shown in Figure 4a.

Behaviors: This swarm captures normal behavior, physical twins of development networks, malicious firmware, faulty/tampered data, and abnormal peripheral control.

Firmware: Details about the normal and anomalous variants of each node are provided in Table III. N_0 sends one byte to all other nodes in the swarm. In its anomalous variant, N_0 generates three random integers (six bytes) in addition to its main function. The anomalous variant doesn't affect other nodes, and its purpose is to detect node-level anomalies with the same network behavior as the normal variant, as well as to check if the GNN misclassifies other nodes due to message passing. N_1 is a sense-type node that generates six floating point numbers (twenty-four bytes) in unique ranges and sends them to N_2. In its anomalous variant, N_1 generates these numbers in an extended range (including the original), creating partially/completely faulty data collection scenarios at downstream nodes (N_2 and N_3). N_2 is a process-type node that receives six floating point numbers (twenty-four bytes) and generates a control signal (six bytes) based on predefined logic. It then sends this processed signal to N_3. In its anomalous variant, N_2 discards the data received from N_1 and instead generates a random control signal for N_3, which affects N_3's operation. Lastly, N_3 is a control-type node that

TABLE V: Normal and anomalous firmware in Swarm-2.

Node	Type	Normal firmware functions	d	Anomalous firmware functions	d
N_0	Control	Broadcasts one byte to all nodes	195	Generate two random integers Broadcasts one byte to all nodes	199
N_1	Sense	Generate four floating point numbers in unique ranges Send data to N_2	438	Generate data in an extended range Send data to N_2	430
N_2	Process	Process received data into a four-byte signal Send processed signal to N_3	490	Generate a random four-byte signal Send control signal to N_3	414
N_3	Control	Control four LEDs using the processed signal	394	Control four LEDs at random	386
N_4	Sense	Generate three floating point numbers in unique ranges Send data to N_5	430	Generate data normally Does not send data to N_5	372
N_5	Process & Control	Process received data into a three-byte signal Control three LEDs using the processed signal	446	Process received data normally Control three unauthorized LEDs	452

TABLE VI: Swarm-2 dataset scenarios.

S.No.	Scenario	Primary Anomaly	Secondary Anomaly	Swarm Label $N_0 - ... - N_5$
1-4	D_{1-4}	-	-	0-0-0-0-0-0
5	AN_0	N_0	-	1-0-0-0-0-0
6	AN_1	N_1	N_2, N_3	0-1-1-1-0-0
7	AN_2	N_2	N_3	0-0-1-1-0-0
8	AN_3	N_3	-	0-0-0-1-0-0
9	AN_4	N_4	N_5	0-0-0-0-1-1
10	AN_5	N_5	-	0-0-0-0-0-1

uses the processed signals from N_2 to control six LEDs. In its anomalous variant, N_3 discards the received signal and instead controls the output LEDs at random.

Dataset: The Swarm-1 dataset includes thirteen scenarios (four normal and nine anomalous), as shown in Table IV. D_1 and D_2 are development sets used for training and testing and are collected from two independent initializations of the original swarm network. In contrast, P_1 and P_2 are collected from two physical twin networks made using a different set of devices. Among the anomalous scenarios, AN_j refers to an anomaly introduced at N_j. Each scenario has 400 synchronized, sequential SRAM traces for each N_j, where each trace is the sequence of integer equivalents of the hex content of the SRAM.

*2) **Swarm-2**:* Swarm-2 is the six-node network shown in Figure 4b that offers a greater challenge to the verifier given its more complex design, a lower number of communicated bytes, a lower degree of change in firmware for anomalous cases, and a larger number of threat types.

Behaviors: This swarm captures normal behavior, malicious firmware, faulty/tampered data, dropped messages, out-of-sync states, abnormal peripheral control, and additional malicious peripherals.

Firmware: The details of the normal and anomalous firmware of each node are given in Table V. N_0 has the same purpose as explained in Swarm-1. It sends one byte to $N_1 - N_5$. In its anomalous variant, N_0 generates two random integers (four bytes). The first branch in this swarm consists of three nodes: N_1 is a process-type node that generates four floating point numbers (sixteen bytes) in unique ranges. Its anomalous variant generates random data in extended ranges, causing downstream effects at N_2 and N_3. N_2 is a process-type node that uses the data received from N_1, and generates a four-byte processed signal for N_3. In its anomalous variant, N_2 sends a random processed signal to N_3, affecting its operation. N_3 is a control-type node that uses the received processed signal to control four LEDs. In its anomalous variant, N_3 generates a random control signal. The second branch of Swarm-2 comprises two nodes: N_4 is a process-type node that generates three floating point numbers (twelve bytes) and sends them to N_5. In its anomalous variant, the function used to send data is commented out (tampering with control dependencies). This anomaly creates an out-of-sync state between N_4 and N_5 and should be detectable at both nodes during attestation. Lastly, N_5 is a process- and control-type node that processes the data received from N_4 and controls three LEDs. In its anomalous variant, N_5 uses the received data to control three LEDs connected to different microcontroller pins.

TABLE VII: Behavior types encapsulated in the swarm datasets and simulated scenarios.

S.No.	Behavior type	Scenarios
1	Normal behavior	D_{1-4}, P_{1-2}
2	Physical twins	P_{1-2}
3	Malicious firmware	$\forall AN_i$
4	Propagated anomalies	AN_1, AN_2, AN_{12}, AN_{13}, AN_4
5	Faulty/tampered data	AN_1, AN_2
6	Abnormal peripheral control	AN_3, AN_5
7	Tampered functions	AN_4
8	Out-of-sync states	AN_4, S_3
9	Added peripherals	AN_5
10	Dropped responses	S_1
11	SRAM perturbation	S_2
12	Trace replay	S_3

Dataset: The Swarm-2 dataset consists of ten scenarios (four normal and six anomalous), as shown in Table VI. $D_1 - D_4$ are development sets for training and testing collected from four separate initializations of the development network. $AN_0 - AN_5$ are six anomalous scenarios where AN_j corresponds to an anomaly at N_j. Each scenario comprises 900 synchronized, sequential SRAM traces for all N_j.

*3) **Simulated Scenarios**:* In addition to the scenarios in the Swarm-1 and Swarm-2 datasets, we use the development sets to simulate the following scenarios:

1. Dropped responses (S_1): Since adversaries may drop attestation responses from the swarm to the gateway, it is

TABLE VIII: Attestation results on Swarm-1 averaged over twenty training and testing phases.

Scenario	GCN				GAT				GT			
	N_0	N_1	N_2	N_3	N_0	N_1	N_2	N_3	N_0	N_1	N_2	N_3
D_1	100	100	100	100	100	100	100	100	100	100	100	100
D_2	100	100	100	100	100	100	100	100	100	100	100	100
P_1	99.67	99.67	100	100	99.67	99.67	100	100	100	100	100	100
P_2	100	99.33	99	100	100	99.33	99	100	100	98.63	99.33	100
AN_0	100	100	97.95	100	100	100	100	100	100	99.98	100	100
AN_1	100	100	13.98	95.38	100	100	3.68	81.87	100	100	97.48	99.02
AN_2	99	100	100	20.72	99.33	100	100	0.83	99.63	100	100	100
AN_3	100	100	99.67	100	100	100	99.67	100	100	100	100	100
AN_{12}	100	100	100	21.48	100	100	100	0.98	100	100	100	100
AN_{23}	99.33	100	100	100	99.67	100	100	100	100	99.97	100	100
AN_{13}	99.78	100	13.45	100	100	100	2.33	100	100	100	96.95	100
AN_{123}	100	100	100	100	100	100	100	100	100	100	100	100
AN_{0123}	100	100	100	100	100	100	100	100	100	100	100	100

TABLE IX: Attestation results on Swarm-2 averaged over twenty training and testing phases.

Scenario	GCN						GAT						GT					
	N_0	N_1	N_2	N_3	N_4	N_5	N_0	N_1	N_2	N_3	N_4	N_5	N_0	N_1	N_2	N_3	N_4	N_5
D_1	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
D_2	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
D_3	100	99.88	100	100	99.88	100	100	99.88	100	100	99.88	100	100	99.88	100	100	100	100
D_4	100	99.75	100	100	100	100	100	99.75	100	100	100	100	100	99.88	100	100	100	100
AN_0	71.03	99.99	100	100	100	100	12.42	99.75	99.94	99.88	100	100	100	99.99	100	100	99.88	100
AN_1	100	100	0	1.12	100	100	100	100	0	2.16	100	100	99.99	100	100	98.08	100	100
AN_2	100	100	100	36.02	100	100	99.97	100	100	41.69	100	100	99.98	99.88	100	100	100	100
AN_3	100	100	100	100	100	100	100	99.88	100	100	100	100	100	100	100	100	100	100
AN_4	99.75	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
AN_5	100	100	100	100	100	100	100	100	100	100	100	100	100	100	99.99	100	100	100

important to evaluate the performance of Algorithm 2 in such a case. To simulate dropped responses, attestation responses from randomly selected N_j are dropped and replaced with the corresponding default trace $t_{def,j}$ from T_{def} during attestation.

2. SRAM perturbation (S_2): An adversary may attempt to forge an SRAM trace or attack the integrity of communication by changing the messages sent from the swarm to the gateway. To simulate such a scenario, we add varying bytes of randomness to the development sets.
3. Trace replay/out-of-sync states (S_3): An adversary may drop messages between nodes or attempt to replay attestation responses collected from nodes. In doing so, they may create network states that are not in sync with the current state of the Sense-Process-Control tasks in the network. To create such a threat scenario, we randomly shuffle the development sets along the temporal axis.

C. Models and Hyperparameters

We experiment with three GNN architectures - GCN, GAT, and GT using the respective *GCNConv*, *GATConv*, and *TransformerConv* layers from PyG. Hyperparameter selection followed an iterative process to select the simplest possible models with the best performance for each GNN architecture to ensure efficacy and efficiency on the verifier. Based on this goal, the best-performing models consisted of two graph layers compressing input data to a latent dimension of 32, followed by a single Linear layer for reconstruction. In all experiments, we use the Adam [48] optimizer with a learning rate of 0.01 and a weight decay ($L2$ regularization on model weights) of 0.0005. Uniform noise sampled from $\mathcal{U}(0,1)$ are added to the inputs with a coefficient of 0.4.

D. Evaluation Metrics

The proposed threshold-based anomaly detection approach is essentially a binary classification problem, wherein similarity scores of SRAM data sections lying above their respective thresholds are flagged as safe (0) and otherwise anomalous (1). We define True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) as the correct 1, correct 0, incorrect 1, and 0 invalid flags after thresholding, respectively.

Subsequently, we use three metrics: Accuracy (A), Detection Rate (DR), and Attestation Rate (AR) defined as follows:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

$$DR = \frac{TP}{TP + FN} \tag{13}$$

$$AR = \frac{TN}{TN + FP} \tag{14}$$

Here, DR and AR are analogous to the true positive and negative rates, respectively.

VII. Results

This section presents the experimental results observed in various test scenarios (categorized in Table VII), an analysis

TABLE X: Comparison with related works.

Method	T_o (s)	Model	Task
Swarm-Net (CPU)	~ 1	GT	Anomaly detection
Swarm-Net (GPU)	~ 1		
WISE [29]	3.5	Crypto, ML	Checksum, clustering
FeSA [30]	~ 1	FL	Classification
Protogerou et. al. [15]	> 5	GNN	Anomaly detection
Rage [31]	$t_c + 0.15$	VGAE	Anomaly detection
Aman et. al. [16]	~ 2	MLP	Classification
HAtt [21]	0.126	Crypto	PUF
SWATT [9]	81	Crypto	Checksum

of performance, and a quantitative comparison with related works.

A. Swarm-1

Table VIII shows the detection results on the Swarm-1 dataset. In addition, Figure 6a compiles these results into four categories: overall performance, authentic firmware, anomalous firmware, and propagated anomalies, and compares the three proposed model architectures.

As the table and figure show, GT has an overall accuracy of 99.83% with a 99.92% AR on authentic firmware, 100% DR on anomalous firmware, and 98.7% DR on propagated anomalies. While GAT and GCN have comparable performance to GT in normal and anomalous firmware samples, their performance is significantly lower on propagated anomalies observed in AN_1, AN_2, AN_{12} and AN_{13}. In such cases, GAT and GCN attain an average DR of 18% and 33%, lowering their overall accuracy to 92.04% and 93.43%, respectively.

B. Swarm-2

Similarly, Table IX and Figure 6b show the results of the three GNN architectures on Swarm-2.

GT achieves an overall accuracy of 99.96% with a 99.99% AR on authentic firmware, 100% DR on anomalous firmware, and 99.52% DR on propagated anomalies. GAT and GCN have comparable AR on authentic firmware (99.98% each); however, their performance is lower on anomalous firmware (specifically on AN_0). As mentioned in Section VI, the anomalous variant of N_0 in Swarm-2 generates four random bytes compared to six in Swarm-1. This makes the detection of the anomaly more difficult. However, GT has a 100% DR in this case. Furthermore, given the reduced number of bytes shared between N_1, N_2 and N_3 compared to Swarm-1, detecting downstream anomalies is a greater challenge in this dataset. GAT and GCN have an average 12.4% and 14.6% DR on propagated anomalies in AN_1 and AN_2. However, they have 100% DR in the case of AN_4. Overall, GT outperforms GCN and GAT in detection capabilities in both datasets, making it the best choice.

C. Simulations

Figure 7 shows *Swarm-Net(GT)*'s performance in the simulated scenarios.

1. Dropped response (S_1): The proposed model has a 99.9% accuracy in attesting swarms when one node's response is dropped at random. It is, therefore, resistant to single points of failure due to dropped messages.
2. Perturbation (S_2): We observe a 95+% DR for 10 or more bytes of random SRAM perturbation.
3. Trace replay/out-of-sync states (S_3): The proposed model achieves an 87.7% DR in detecting trace replay attacks and out-of-sync network states. Thus, attempts at such threats are likely to be detected.

D. Intuition Behind Model Architectures

Since IoT swarms have a graph-like structure, GNNs are an obvious choice - among which GT, GCN, and GAT are the three most popular GNN architectures in the literature due to their ability to handle graph-structured data efficiently using different aggregation functions. Based on our discussion in Section III on the efficacy of their aggregation methods for modeling complex inter-node relationships in IoT networks, GT has an advantage over GAT and GCN due to the higher degree of parameterization in its design compared to GAT and the appropriate modeling of edges compared to the oversimplified averaging in GCN, which may not correctly represent relationships between the SRAM contents of connected nodes. This is supported by the results observed for Swarm-1, Swarm-2, and the simulated scenarios.

E. Latency and Memory

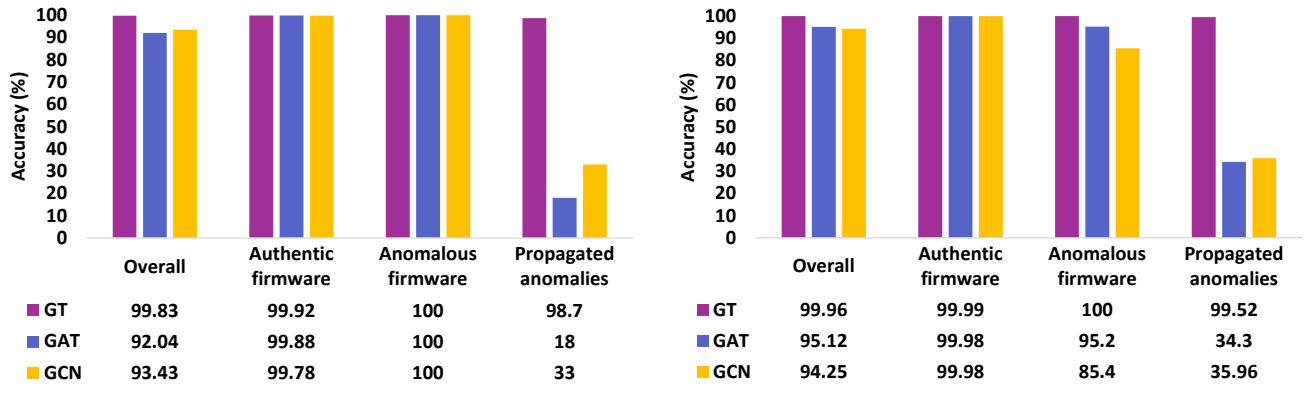
Overhead (T_o) may be defined as the summation of the communication overhead (t_c), preprocessing (t_p), and inference time (t_i) following the equation:

$$T_o = t_c + t_p + t_i \tag{15}$$

Table X compares latency between various related works. Since *Swarm-Net* is the first paper to use SRAM for swarm attestation, it differs vastly from the related works in its nature, and the methods used in other studies may not apply to SRAM-based analysis. Thus, our comparison is limited to the total attestation overhead on the verifier's end and computation complexity on the IoT devices.

As the table shows, *Swarm-Net* has a $T_o = 1$ seconds, most of which is associated with the data collection time. The processing time on the IoT devices (included in t_c) is low ($\sim$ milliseconds) since they need not perform complex computation on the SRAM traces, unlike other software-based RA methods [9, 10, 22]. In addition, the proposed method uses a single SRAM trace for attestation, compared to WISE [15], which requires over 5 seconds of network flow information for attestation. It is worth noting that the communication overhead stays constant for larger swarms since the gateway sends attestation requests in parallel and collects the prover responses asynchronously.

The evaluation time ($t_p + t_i$) of our best model (GT) is of the order 10^{-5} for both CPU- and GPU-based verifiers, which can be attributed to its simple design (compared to the more complex VGAE proposed in RAGE [31] designed for single-node attestation). Lastly, the verifier needs around 10 minutes



(a) Swarm-1 (b) Swarm-2

Fig. 6: Performance of the proposed graph models across different detection tasks.

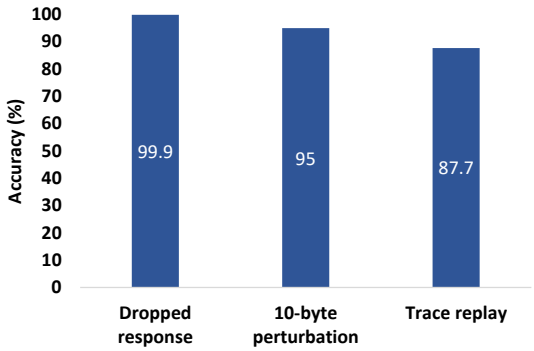


Fig. 7: Performance of *Swarm-Net(GT)* on simulated attacks.

to sample enough traces for training the GNN and incurs 0.036 seconds (GPU) and 0.563 seconds (CPU) per epoch (for GT) during the training phase.

F. Scalability

The proposed GT model consists of $600,448$ trainable parameters and occupies ~ 2.4 MB of verifier memory per swarm. To examine the viability of *Swarm-Net* on large network sizes, we simulated a 1000-node GT and tested it on randomized input traces. The model occupied 400MB of verifier memory and incurred 10^{-2} seconds of evaluation latency per swarm response. Furthermore, since the gateway collects SRAM responses asynchronously, the data collection time depends on the timeout duration (1 second in this study) and would be the same for large IoT networks. Furthermore, testing the proposed method's overall attestation performance on swarms of such a scale is impossible, given the practical difficulty of creating meaningful physical networks and firmware. However, given its efficacy on the collected and simulated dataset, we expect *Swarm-Net* to perform comparably well on larger swarms.

VIII. Security Analysis

We now present a security analysis of the *Swarm-Net* attestation protocol.

Lemma 1. *Consistency: The data sections obtained from the same firmware on physical twins behave similarly.*

The same firmware loaded on two different devices with the same functionality can generate data sections following similar behavior. The results of the proposed attestation algorithm on P_1 and P_2 from the Swarm-1 dataset support this claim.

Lemma 2. *Distinguishability: Only an authorized firmware can generate an authentic SRAM trace using its own SRAM.*

Changes made to the variables and control dependencies in the firmware create observable differences in the SRAM behavior, as supported by the results on node-level anomalies AN_j in Section VII. Thus, an unauthorized firmware cannot generate a valid T' using its own SRAM.

Theorem 3. *Mutual Authentication: Completing one protocol run implies that the verifier, gateway, and IoT node have done so with a legitimate counterpart.*

Proof. Since the communication between the verifier and gateway is assumed to be secure, an adversary may only impersonate either (1) the trusted gateway or (2) an IoT node N_j.

In case (1), the adversary must furnish a valid parameter $I_{req,j}$ and subsequently $I_{update,j}$ and $m_{update,j}$ to N_j, which is not possible without the knowledge of the shared secret key, K_j, and the nonce for that attestation round, C_j.

In case (2), the adversary must furnish a valid parameter $I_{resp,j}$ and message $m_{resp,j}$ to the gateway, which is not possible without knowledge of the shared secret key, K_j, and the updated nonce for the attestation round, C_j. □

Theorem 4. *Availability: A registered swarm is always available.*

Proof. To affect the availability of a registered swarm, an adversary may do one of three things: (1) denial of service on the swarm, (2) replay old attestation requests from the gateway, or (3) drop messages between the gateway and the swarm.

In case (1), an adversary may launch a denial of service on the swarm by sending frequent authentication requests to the swarm, which is not possible without the knowledge of K_j and C_j between the gateway and each N_j in the swarm.

In case (2), an adversary may replay previously valid attestation parameters $I_{req,j}$ to each N_j in the swarm. However, these parameters will not be valid for subsequent attestation rounds as the nonce C_j is updated by the gateway and the swarm nodes after every attestation round.

Lastly, in case (3), an adversary may drop messages between the gateway and the swarm, causing a state of desynchronization. However, the gateway maintains a list of valid nonces L_R as mentioned in Section V, which it uses to resynchronize the devices in future attestation rounds. Furthermore, the verifier maintains a list of default traces T_{def}, which it uses to replace missing responses during attestation, thereby avoiding single points of failure as shown by the results on dropped responses in Section VII-C. □

Theorem 5. *Attestation: Successful attestation by Algorithm 2 proves that the provers have authentic firmware.*

Proof. An adversary may attempt the following: (1) generate an authentic trace from the SRAM of a device loaded with malicious firmware, and (2) attempt to capture and replay a valid trace.

Case (1) is impossible, as stated in Lemma 2; malicious firmware cannot generate a valid trace using the microcontroller's SRAM. Furthermore, traces generated from malicious firmware are easily detected by Algorithm 2 as supported by the results in Section VII. Finally, an adversary may attempt to forge SRAM traces using a random guess. Given a maximum padding length L and minimum m bytes of detectable uniform random perturbation by the attestation method, the adversary's probability (P_{Adv}) of fooling the attestation method is as follows.

$$P_{Adv} \leq \frac{m}{L} \tag{16}$$

Based on our perturbation experiments on S_2 in Section VII-C, *Swarm-Net* can detect $m = 10$ bytes or more of uniform random noise for $L = 2048$. Thus, the adversary has a $P_{Adv} \leq 0.0048$ in such an attack, provided it can bypass the confidentiality and integrity measures in addition to forging an SRAM trace.

In case (2), the adversary cannot access the valid traces since the communication between the verifier and the swarm is encrypted. However, even if an adversary succeeds in doing so, replaying old traces will create out-of-sync network states, which are detectable during attestation based on the results of trace replay attacks in Section VII-C.

□

IX. Conclusion

This paper proposed the first SRAM-based swarm attestation approach, *Swarm-Net*, that exploited the graph-like structure of IoT swarms using GNNs. It presented the first datasets on SRAM-based attestation that cover various complicated node-level and inter-node relationships. In addition, a secure protocol was proposed that ensured confidentiality, integrity, mutual authentication, and attestation. *Swarm-Net* achieved a 99.9% overall accuracy across all types of behaviors ranging from normal firmware to anomalous firmware and propagated anomalies. It was also tested on simulated scenarios, such as dropped responses, trace replay attacks, and SRAM perturbation, which showed resistance to such attacks. Latency evaluation showed an overhead and evaluation latency of the order of 1 second and 10^{-5} seconds, respectively. Lastly, a security analysis highlighted security against impersonation, replay attacks, denial of service, dropped messages, malicious firmware, and propagated anomalies. Future studies can attempt swarm attestation using a smaller number of data section bytes. Furthermore, while this study uses single SRAM responses for attestation, time-series information captured by sequences of SRAM traces may help detect intermittent malicious activity and can be explored by future studies.

References

[1] V. Chamola, V. Hassija, V. Gupta, and M. Guizani, "A comprehensive review of the covid-19 pandemic and the role of iot, drones, ai, blockchain, and 5g in managing its impact," *Ieee access*, vol. 8, pp. 90 225–90 265, 2020.

[2] B. Marr. 2024 iot and smart device trends: What you need to know for the future. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2023/10/19/2024-iot-and-smart-device-trends-what-you-need-to-know-for-the-future/

[3] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on iot security: application areas, security threats, and solution architectures," *IEEe Access*, vol. 7, pp. 82 721–82 743, 2019.

[4] L. Ilascu, "When their firmware is vulnerable, its up to you to protect your smart devices," *Accessed: May*, vol. 5, 2019.

[5] S. F. J. J. Ankergård, E. Dushku, and N. Dragoni, "State-of-the-art software-based remote attestation: Opportunities and open issues for internet of things," *Sensors*, vol. 21, no. 5, p. 1598, 2021.

[6] I. Sfyrakis and T. Gross, "A survey on hardware approaches for remote attestation in network infrastructures," *arXiv preprint arXiv:2005.12453*, 2020.

[7] W. A. Johnson, S. Ghafoor, and S. Prowell, "A taxonomy and review of remote attestation schemes in embedded systems," *IEEE Access*, vol. 9, pp. 142 390–142 410, 2021.

[8] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, "Collective remote attestation at the internet of things scale: State-of-the-art and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2447–2461, 2020.

[9] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, "Swatt: Software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004.* IEEE, 2004, pp. 272–282.

[10] A. Seshadri, M. Luk, A. Perrig, L. Van Doorn, and P. Khosla, "Scuba: Secure code update by attestation in sensor networks," in *Proceedings of the 5th ACM workshop on Wireless security*, 2006, pp. 85–94.

[11] A. Seshadri, M. Luk, and A. Perrig, "Sake: Software attestation for key establishment in sensor networks," in *Distributed Computing in Sensor Systems: 4th IEEE International Conference, DCOSS 2008 Santorini Island, Greece, June 11-14, 2008 Proceedings 4*. Springer, 2008, pp. 372–385.

[12] S. Agrawal, M. L. Das, A. Mathuria, and S. Srivastava, "Program integrity verification for detecting node capture attack in wireless sensor network," in *Information Systems Security: 11th International Conference, ICISS 2015, Kolkata, India, December 16-20, 2015. Proceedings 11*. Springer, 2015, pp. 419–440.

[13] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: Tiny trust anchor for tiny devices," in *Proceedings of the 52nd annual design automation conference*, 2015, pp. 1–6.

[14] S. W. Kibret, "Property-based attestation in device swarms: a machine learning approach," *Machine Learning for Cyber Security*, p. 71, 2023.

[15] A. Protogerou, S. Papadopoulos, A. Drosou, D. Tzovaras, and I. Refanidis, "A graph neural network method for distributed anomaly detection in iot," *Evolving Systems*, vol. 12, no. 1, pp. 19–36, 2021.

[16] M. N. Aman, H. Basheer, J. W. Wong, J. Xu, H. W. Lim, and B. Sikdar, "Machine-learning-based attestation for the internet of things using memory traces," *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 20 431–20 443, 2022.

[17] C. Krauß, F. Stumpf, and C. Eckert, "Detecting node compromise in hybrid wireless sensor networks using attestation techniques," in *Security and Privacy in Ad-hoc and Sensor Networks: 4th European Workshop, ESAS 2007, Cambridge, UK, July 2-3, 2007. Proceedings 4*. Springer, 2007, pp. 203–217.

[18] H. Tan, W. Hu, and S. Jha, "A tpm-enabled remote attestation protocol (trap) in wireless sensor networks," in *Proceedings of the 6th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, 2011, pp. 9–16.

[19] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "Smart: secure and minimal architecture for (establishing dynamic) root of trust." in *Ndss*, vol. 12, 2012, pp. 1–15.

[20] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems*, 2014, pp. 1–14.

[21] M. N. Aman, M. H. Basheer, S. Dash, J. W. Wong, J. Xu, H. W. Lim, and B. Sikdar, "Hatt: Hybrid remote attestation for the internet of things with high availability," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7220–7233, 2020.

[22] M. Khodari, A. Rawat, M. Asplund, and A. Gurtov, "Decentralized firmware attestation for in-vehicle networks," in *Proceedings of the 5th on Cyber-Physical System Security Workshop*, 2019, pp. 47–56.

[23] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "Seda: Scalable embedded device attestation," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 964–975.

[24] X. Carpent, K. ElDefrawy, N. Rattanavipanon, and G. Tsudik, "Lightweight swarm attestation: A tale of two lisa-s," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 86–100.

[25] A. Visintin, F. Toffalini, M. Conti, and J. Zhou, "Safeˆ d: Self-attestation for networks of heterogeneous embedded devices," *arXiv preprint arXiv:1909.08168*, 2019.

[26] B. Kuang, A. Fu, S. Yu, G. Yang, M. Su, and Y. Zhang, "Esdra: An efficient and secure distributed remote attestation scheme for iot swarms," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8372–8383, 2019.

[27] E. Dushku, M. M. Rabbani, M. Conti, L. V. Mancini, and S. Ranise, "Sara: Secure asynchronous remote attestation for iot systems," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3123–3136, 2020.

[28] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, "Healed: Healing & attestation for low-end embedded devices," in *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer, 2019, pp. 627–645.

[29] M. Ammar, M. Washha, and B. Crispo, "Wise: Lightweight intelligent swarm attestation scheme for iot (the verifier's perspective)," in *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2018, pp. 1–8.

[30] B. Kuang, A. Fu, Y. Gao, Y. Zhang, J. Zhou, and R. H. Deng, "Fesa: Automatic federated swarm attestation on dynamic large-scale iot devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2954–2969, 2022.

[31] M. Chilese, R. Mitev, M. Orenbach, R. Thorburn, A. Atamli, and A.-R. Sadeghi, "One for all and all for one: Gnn-based control-flow attestation for embedded devices," *arXiv preprint arXiv:2403.07465*, 2024.

[32] M. Jakobsson and K.-A. Johansson, "Retroactive detection of malware with applications to mobile platforms," in *5th USENIX Workshop on Hot Topics in Security (HotSec 10)*, 2010.

[33] Y. Li, J. M. McCune, and A. Perrig, "Sbap: Software-based attestation for peripherals," in *Trust and Trustworthy Computing: Third International Conference, TRUST 2010, Berlin, Germany, June 21-23, 2010. Proceedings 3*. Springer, 2010, pp. 16–29.

[34] B. Chen, X. Dong, G. Bai, S. Jauhar, and Y. Cheng, "Secure and efficient software-based attestation for industrial control devices with arm processors," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 425–436.

[35] M. N. Aman and B. Sikdar, "Att-auth: A hybrid protocol for industrial iot attestation with authentication," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5119–5131, 2018.

[36] S. Hristozov, J. Heyszl, S. Wagner, and G. Sigl, "Practical runtime attestation for tiny iot devices," in *NDSS Workshop on Decentralized IoT Security and Standards (DISS)*, vol. 18, 2018.

[37] J. Koshy and R. Pandey, "Remote incremental linking for energy-efficient reprogramming of sensor networks," in *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.* IEEE, 2005, pp. 354–365.

[38] V. Kohli, M. N. Aman, and B. Sikdar, "An intelligent fingerprinting technique for low-power embedded iot devices," *IEEE Transactions on Artificial Intelligence*, 2024.

[39] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[40] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[41] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[42] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," *arXiv preprint arXiv:2009.03509*, 2020.

[43] P. Xu, X. Zhu, and D. A. Clifton, "Multimodal learning with transformers: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 10, pp. 12 113–12 132, 2023.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[45] F. Samiullah, M. L. Gan, S. Akleylek, and Y. Aun, "Group key management in internet of things: A systematic literature review," *IEEE Access*, 2023.

[46] H. Gilbert and H. Handschuh, "Security analysis of sha-256 and sisters," in *International workshop on selected areas in cryptography*. Springer, 2003, pp. 175–193.

[47] V. Kohli, B. Kohli, M. Naveed Aman, and B. Sikdar, "Iot device swarm sram dataset for firmware attestation," 2024. [Online]. Available: https://dx.doi.org/10.21227/gmee-vj41

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

Varun Kohli is a Ph.D. student at the Department of Electrical and Computer Engineering at the National University of Singapore. He received his B.E. in Electrical and Electronics Engineering from the Birla Institute of Technology and Science, Pilani, India, in 2021. His research interests include Artificial Intelligence, IoT, and Security.

Bhavya Kohli is an undergraduate student at IIT Bombay, India. He is pursuing his B.Tech in Electrical Engineering and M.Tech in Artificial Intelligence as an interdisciplinary dual degree, with an expected graduation in 2025. His research interests include Machine Learning, Security, and real-world applications of ML.

Muhammad Naveed Aman an Assistant Professor in the University of Nebraska-Lincoln received the B.Sc. degree in Computer Systems Engineering from KPK UET, Peshawar, Pakistan, M.Sc. degree in Computer Engineering from the Center for Advanced Studies in Engineering, Islamabad, Pakistan, M.Engg. degree in Industrial and Management Engineering and Ph.D. in Electrical Engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA in 2006, 2008, and 2012, respectively. His research interests include IoT security.

Biplab Sikdar received the B.Tech. degree in electronics and communication engineering from North Eastern Hill University, Shillong, India, in 1996, the M.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1998, and the Ph.D. degree in electrical engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA, in 2001. He is currently a Professor with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. His research interests include wireless network, and security for IoT and cyber-physical systems.