ISSN: 2093-5374 / E-ISSN: 2093-5382

VADIA-Verkle Tree-based Approach for Dealing Data Integrity Attacks in Opportunistic Mobile Social Networks

Vimitha R Vidhya Lakshmi[1*], and Dr.T. Gireesh Kumar[2]

[1*]TIFAC-CORE in Cyber Security, Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India. vimirsvv@gmail.com, https://orcid.org/0000-0002-6152-2780

[2]Department of Computer Science and Engineering, Amrita School of Computing, Coimbatore, Amrita Vishwa Vidyapeetham, India. t_gireeshkumar@cb.amrita.edu, https://orcid.org/0000-0002-6313-3850

Received: September 13, 2023; Revised: November 16, 2023; Accepted: January 16, 2024; Published: March 30, 2024

Abstract

Opportunistic Mobile Social Networks (OMSN) are prone to data integrity attacks that jeopardize the integrity of the routing data inside the network. Among the several techniques that cope with these attacks in OMSN, tree-based approaches have proven to be the most effective due to its ease of data verification and ensurance in data integrity. This paper evaluates two tree-based data structures, Merkle tree and Verkle tree in terms of their effectiveness in detecting and preventing such attacks. The evaluation considers tree-generation time and proof-checking time, and the results demonstrate that the Verkle tree is a bandwidth-efficient solution and have lower proof-checking time, with a reduction of 98.33% than Merkle tree. This makes Verkle tree a good choice for handling data integrity attacks in OMSN. A Verkle tree-based approach, named VADIA, is proposed to handle data integrity attacks such as packet dropping, packet modification and pollution attack in OMSN. The proposed approach is implemented in the Opportunistic Network Environment (ONE) simulator and is shown to be effective in detecting malicious nodes and paths, reducing false negative rates, and improving accuracy in detecting malicious activities. The results demonstrate a 47%, 84% and 69% improvement in malicious node, malicious path and malicious activity detection over a period of time. Furthermore, the approach achieves an 80% reduction in false negative rates.

Keywords: Data Integrity, Malicious Node, Merkle Tree, Opportunistic Networks, Security, Verkle Tree.

1 Introduction

One of the most interesting and booming technology in mobile communication is Opportunistic network (OppNet). OppNet falls under Delay Tolerant Network (DTN) (Fall et al., 2003), which is also referred as challenged network because of its special characteristics like no end-end path for message transmission, asymmetric data rates, high delay and high error rates. An OppNet which utilize mobile nodes along with the human social characteristics of the nodes to exchange information among each other is known as Opportunistic Mobile Social Network (OMSN) (Lakshmi & Gireesh, 2019). OMSN is a type of Mobile Social Network (Lakshmi & Gireesh, 2017), that operates on any short-range technologies like Bluetooth, Direct Wi-Fi or Near Field Communication (NFC).

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), volume: 15, number: 1 (March), pp. 154-171. DOI: *10.58346/JOWUA.2024.I1.011*

*Corresponding author: TIFAC-CORE in Cyber Security, Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India.

OMSN security is more important than any other networks, since the OMSN users are strangers to each other and the possibility of attack occurrences are very high (Kaur, M., & Mahajan, M., 2013). OMSN is vulnerable to many kinds of attacks especially the attacks that occurs while routing. Routing/data integrity attacks pose a threat to the accuracy and consistency of the information transmitted between nodes in an OMSN (Sonya, A., & Kavitha, G., 2022). Such attacks are carried out by malicious nodes with the intent of altering the messages being transmitted, potentially leading to erroneous decisions and negative impacts on the network (Pieters, W., 2011). In networks, especially in challenged networks, data integrity attacks are considered quite difficult to be identified and prevented. Most common data integrity attacks are:

- Packet Dropping or selective Packet Dropping Attack (PDA): The malicious node/attacker selectively drops some random packets, not all, when the packet is transmitted between two mobile nodes (Kang, M., 2020).
- Packet Modification Attack (PMA) is divided into selective PMA, where attacker modifies some packets and complete PMA, where attacker modifies all the packets that is transmitted between the nodes.
- Pollution Attack (PA): Also known as packet injection/packet faking attack. The attacker drops some (not all) packets and inject new packets instead of them.

Merkle tree is one of the most recent and existing data structure utilized in OMSN for checking data integrity attacks. Many recent works provide binary merkle tree-based solution (Alajeely, Doss & Mak-Hau, 2017; Alajeely & Doss, 2019) for handling data integrity attacks in OMSN. However, a notable limitation of this approach is the large size of the merkle proof required for verifying the integrity of a message. As the number of packets increases, the merkle tree grows and merkle proof increases exponentially. The large size of the merkle proof can result in significant network bandwidth overhead, making it a costly solution. Verkle tree (Kuszmaul, 2019) is a relatively new data structure that has gained attention for its efficacy in addressing data integrity attacks, owing to its ease of use in data validation and verification. This data structure emerged as an alternative to merkle tree, due to its smaller proof size.

The main contributions of this paper are:

- **Merkle tree and Verkle tree Comparison**: Designed and implemented two algorithms: *Verkle Tree Generation* and *Verkle Tree Proof_Checking*. The performance of Merkle tree is compared with verkle tree in terms of tree generation time and proof checking time.
- **VADIA**: A verkle tree-based solution is designed as a mitigation strategy for dealing with data integrity attacks in OMSN. The attacks focused are PDA, PMA, and PA.

This paper is organized as follows: Section 2 is about the related works that exists in this area. Section 3 is about the proposed work and the methodology used. Section 4 is the experimental settings and results obtained from the proposed methodology. Finally, section 5 concludes the paper and presents the future work.

2 Related Works

Mitigation strategies for data integrity attacks seen in opportunistic networks are listed in Table 1, along with their limitations. It is clear that, binary merkle tree-based solution contribute a secure and complete solution for dealing data integrity attacks in OMSN. The only limitation observed is its proof size.

Table 1: Existing Works based on Data Integrity Attacks in Opportunistic Networks

Author and Year	Category	Attack Demonstrated	Methodology	Limitations
Alajeely & Doss (2014)	Algorithm-based	Packet dropping attack	• Modifies the packet header and utilizes indicative field to detect the attack and detects malicious node that cause the attack • Attack detection by intermediate nodes	• Lack of security
Alajeely & Doss (2015a)	Binary merkle tree-based	Pollution/Packet faking attack	• Utilizes merkle tree hashing technique to detect the attack and detects malicious node that cause the attack	• Merkle proof size is very large, so not bandwidth efficient
Alajeely & Doss (2015b)	Cryptography-based	Pollution/Packet faking attack	• Utilizes hash chain technique to detect the attack and detects malicious node that cause the attack	• Malicious path is not detected
Alajeely, Ahmad, Doss & Mak-Hau (2015)	Algorithm-based	Pollution/Packet faking attack	• Utilizes packet creation time to detect the attack • Attack detection by intermediate nodes	• Lack of security • Malicious node and path not detected
Alajeely & Doss (2016)	Binary merkle tree, Trust and Reputation-based	Packet dropping attack	• Utilizes merkle tree hashing technique to detect the attack and detects malicious node and malicious path that cause the attack • Utilizes direct and indirect trust to calculate reputation in-order to detect malicious nodes	• Merkle proof size is very large, so not bandwidth efficient
Rashidibajgan (2017)	Game theory and Trust-based	Packet dropping and selfish attack	• Nash equilibrium methodology is used to detect the attack • Nodes generate private and public keys • Trust is calculated for data forwarding	• Works inside a small area only
Ahmad, Doss, Alajeely & Rubeaai (2018)	Binary merkle Tree and Trust-based	Packet dropping attack	• Utilizes merkle tree hashing technique to detect the attack • Detection of malicious path and malicious node that cause the attack • Utilizes trust values to detect malicious nodes	• Malicious node and attack are detected by destination node alone • Merkle proof size is very large, so not bandwidth efficient
Doss, Alajeely & Rubeaai (2018)	Binary merkle tree, Trust and Reputation-based	On/off packet modification attack	• Utilizes merkle tree hashing technique to detect the attack • Utilizes direct, meeting and indirect trust to calculate reputation in-order to detect malicious nodes and malicious path	• Merkle proof size is very large, so not bandwidth efficient
Gupta, (2019)	Cryptography-based	Packet modification and replay attack	• Cyclic Redundancy Check methodology is utilized to detect the attack	• Malicious node and path not detected • Attack is detected by destination node alone
Wang & Kazuya (2020)	Cryptography and trust-based	Packet dropping, tapping, replay, sybil and man-in-the-middle attack	• New authentication architecture using identity-based encryption is introduced • Distributed key generation centres are introduced and are enabled by trust framework • Scheme is light-weight and its security is proven by indistinguishable-based privacy model using random oracles	• Malicious nodes are prevented and not detected • Malicious path is not detected • Difficulties in key maintenance
Rashidibajgan, Hupperich, Doss & Förster, (2021)	Cryptography and trust-based	Packet dropping, sybil and selfish attack	• Utilizes public key sharing, trust-based and node's cooperation algorithms • Privacy-preserving history-based routing algorithm is proposed	• Performance of the algorithm decreases as the number of malicious nodes increases • Malicious path is not detected • Difficulties in key maintenance
Altaweel, Stoleru, Gu, Maity & Bhunia (2022)	Algorithm-based	Collusive Hijack attack: packet modification, traffic analysis, and incentive seeking attacks	• The attack is identified by the Kolmogorov-Smirnov two sample test • Path, hop and early hop detection techniques are utilized to detect the attack • Have high detection rates with low false positive rate and detection latency	• Malicious path not detected

Problem Formulation: The literature survey suggests that the state-of-the-art technique for detecting and preventing malicious nodes during routing in OMSN is based on the merkle tree. The merkle tree provides a constant-sized digest and enables efficient data integrity and validation using merkle proofs. However, a binary Merkle tree with a large number of leaf nodes can result in larger proof sizes and higher bandwidth overhead, which can be costly. To address this issue, the *K*-ary merkle tree was proposed as a solution to decrease the depth of binary merkle tree (*K* is the branching factor). This reduced the height of the merkle tree from $\log_2 n$ to $\log_K n$. However, the merkle proof size grew from $O(\log_2 n)$ to $O(K\log_K n)$, as the proof considered (*K*-1) nodes at each level. As a potential solution to the larger proof size problem, the verkle tree was introduced as a bandwidth-efficient alternative to the merkle tree. The verkle tree is speculated to be well-suited for dealing with data integrity attacks in OMSN. *K*-ary verkle tree have proof size of $O(\log_2 K)$ times smaller than the merkle tree, without much compromising construction time (Kuszmaul, 2019).

The state-of-the-art technique (merkle tree) is compared with verkle tree for determining which data structure is better in terms of tree generation time, proof checking time and proof size. And then with the best-performing data structure the malicious node is detected and prevented in OMSN.

3 Methodology

Merkle Tree

Merkle tree (hash tree) (Merkle, 1980) is a special kind of tree structure, in which the leaf nodes are the hash values of the data and every non-leaf node are the XORed hash values of its child nodes ($h(p_{xy})=h(p_x)\oplus h(p_y)$). OMSN utilizes this tree structure to analyze the integrity of the data, that is being transmitted among the nodes in the network.

Working principle of merkle tree: If a sender node *X* wants to send a data *D* to destination node *Y*. Initially, *D* is divided into *n* number of packets ($D = \{p_1, p_2, \ldots p_n\}$). Instead of storing *n* hashes, *X* constructs a merkle tree and the root value *R* is appended in each p_i's header, which is transmitted to *Y*. *Y* on receiving all the p_i, re-constructs the merkle tree and calculates new root value *R'*. *Y* then matches *R* and *R'*, if both the values are equal then data integrity is maintained during the transmission ($X \rightarrow Y$).

For checking the integrity of individual packet, merkle proof is utilized. Figure. 1 illustrates binary merkle tree for 4 packets and the shaded nodes ($h(p_2)$, $h(p_{01})$) are the merkle proof for the packet p_3. If *Y* needs to check the integrity of packet p_3, *X* sends p_3 along with its merkle proof and root value *R*. This merkle proof is sufficient to determine the new root value *R'*. *Y* then matches *R* and *R'*, if both the values are equal then data integrity is maintained for the packet p_3. Figure. 2 illustrates *K*-ary merkle tree (3-ary merkle tree) and the shaded nodes ($h(p_4)$, $h(p_5)$, $h(p_{012})$, $h(p_{678})$) are the merkle proof for the packet p_3. The merkle proof size grows in proportion to the *K* value.

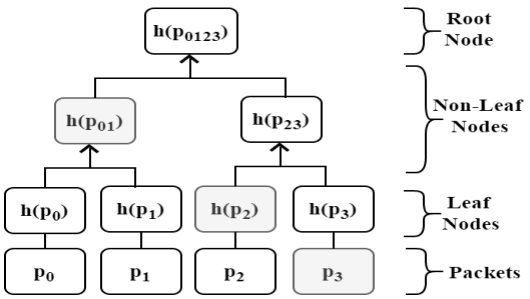


Figure 1: Binary Merkle Tree with Merkle Proof for p3 (Shaded Nodes)

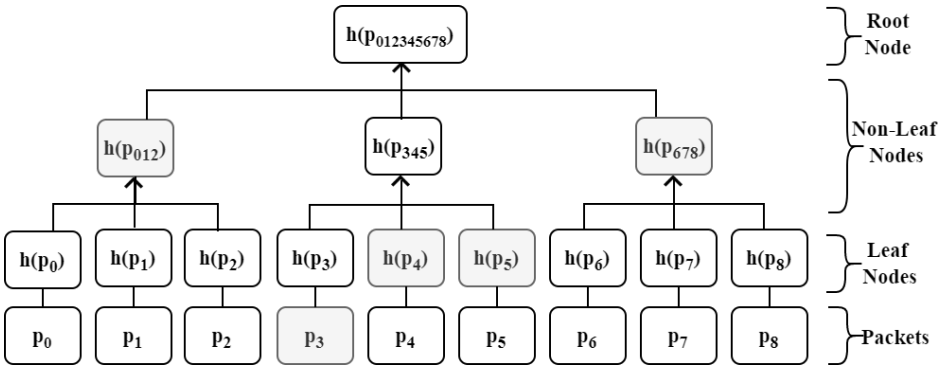


Figure 2: K-ary Merkle Tree (K = 3) with Merkle Proof for p3 (Shaded Nodes)

Verkle Tree

Verkle tree is a sort of merkle tree, that are constructed using vector commitments (*VC*) and membership proofs (*Л*) rather than conventional hash functions. The vector commitment scheme utilized is based on Computational Diffie-Hellman assumption (Catalano & Fiore, 2013). Figure. 3 depicts verkle tree with $K = 2$, along with *VC* and *Л* values. For example, suppose a data (*D*) is divided into four data packets (p_0, p_1, p_2, p_3), then VC_0, VC_1, VC_2 are the vector commitments, $Л_0$, $Л_1$, $Л_2$, $Л_3$, $Л_4$, $Л_5$ are the membership proofs and VC_2 is the Root Commitment (*RC*).

Working principle of verkle tree: If a sender node *X* wants to send a data *D* to destination node *Y*. Initially, *D* is divided into *n* number of packets ($D = \{p_1, p_2, \ldots p_n\}$) and *X* constructs a verkle tree by generating a verkle key. This verkle key along with the *R*C is appended in each p_i's header, which is then transmitted to *Y*. *Y* on receiving all the p_i, re-constructs the verkle tree and calculates *RC'*. *Y* then matches *RC* and *RC'*, if both the values are equal then data integrity is maintained during transmission ($X \rightarrow Y$).

For checking the integrity of individual packet, the *Verkle Tree Proof_Checking* algorithm is utilized. The verkle proof is an important part of this algorithm. Verkle proof comprises of the membership proof value of the specific packet to be checked as well as all the membership proof values along its path upto the root of the verkle tree. For example, from figure. 3, the verkle proof for the packet p_3 consists of $Л_3$ and $Л_5$. If *Y* needs to check the integrity of packet p_3, *X* sends all the data packets along with its *RC* value. Y on receiving all the data packets, calculates the verkle proof and its indexes. Here, for the verkle proof ($Л_3$, $Л_5$), the verkle proof indexes are (*3*, *5*). This verkle proof indexes are sufficient to determine the *RC'* value. *Y* then matches *RC* and *RC'*, if both the values are equal then data integrity is maintained for the packet p_3.

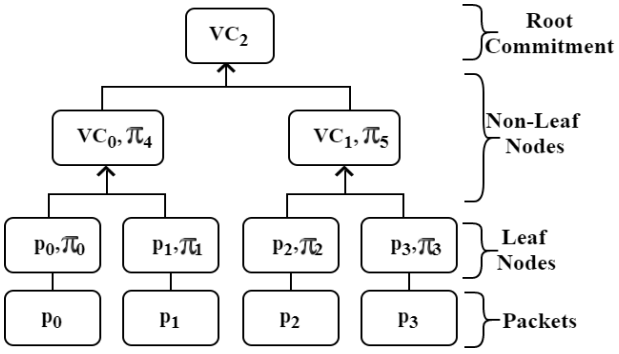


Figure 3: K-ary Verkle Tree with K = 2

Algorithm for Verkle Tree Generation and Proof Checking

Let G, G_T be two bilinear groups with p as prime order and e as bilinear mapping

e: $G \times G \rightarrow G_T$

$K \rightarrow$ Branching Factor; $K \in \mathbb{N}$

$D \rightarrow$ data

$q \rightarrow$ number of data packets (D is divided into q number of data packets)

$g \rightarrow$ random generator; $g \in G$

$r \rightarrow$ number of subsets in the verkle tree; $r = \lceil q/K \rceil$

$l \rightarrow [l_0, l_1, l_2, \ldots, l_{r-1}]$; number of packets in each subset s

For $i = 0$ to $r - 1$

$l_i = K$; number of packets in subset i

If $(q - (r-1)K) \neq K$

$l_{r-1} = q - (r-1)K$

$\mathbb{Z}_p \rightarrow [z_0, z_1, z_2, \ldots, z_{q-1}]$ (Randomly chosen)

$\forall i = 0,1,2, \ldots, q\text{-}1;\ h_i = g^{z_i}$

$\forall i, j = 0,1,2, \ldots, q\text{-}1;\ h_{i,j} = g^{z_i z_j}\ (i \neq j)$

$p \rightarrow [p_0, p_1, p_2, \ldots, p_{q-1}]$; data packet

$HT \rightarrow$ Height of the tree; $HT = \lceil \log_K(q) \rceil$

Number of levels in the verkle tree $= HT + 1$

$rs \rightarrow [rs_0, rs_1, rs_2, \ldots, rs_{HT}]$; number of subsets at each level of the verkle tree

$rs_0 = r$

For $i = 1$ to HT

$rs_i = \left\lceil {}^{rs_{i-1}}/_{K} \right\rceil$

$VC \rightarrow [VC_0, VC_1, VC_2, \ldots, VC_{|q-K+1|}]$; Vector Commitment

$trs \rightarrow$ total number of subsets in tree without considering root

For $i = 0$ to $HT\text{-}1$

$trs = trs + rs[i]$

$Л \rightarrow [Л_0, Л_1, Л_2, \ldots, Л_{q+trs+1}]$; membership proof

$ri \rightarrow$ root index; Initialize: $ri = 0$

$Root_{VC} \rightarrow$ root vector commitment

$k \rightarrow$ key; $k = (g, \{h_i\}_{i \in [q-1]}, \{h_{i,j}\}_{i,j \in [q-1], i \neq j})$

$VP \rightarrow [VP_0, VP_1, VP_2, \ldots, VP_{(HT+1)}]$; Verkle Proof indexes

$nVC \rightarrow [nVC_0, nVC_1, nVC_2, \ldots, nVC_{|q-K+1|}]$; new Vector Commitment

Assumptions: Either consider complete verkle tree upto height 2 or consider perfect verkle tree ($q = K^{HT}$) of all heights.

Algorithm ***Verkle Tree Generation*** (K, q, k, p_i, r, l_i, HT, rs_i)

1. If ($HT == 1$), skip steps 4 and 5
2. Compute VC over each s

   For $n = 0$ to $r - 1$

   $$VC_n = \prod_{i=n.K}^{t} h_i^{p_i} \quad \text{where, } t = \begin{cases} n.l_n + (K-1); & l_n = K \\ n.K + l_n - 1 & ; \ l_n < K \end{cases}$$

   $ri = n$
3. Compute $Л$ for each data packet with respect to VC

   For $n = 0$ to $r - 1$

For $i = n.K$ to t where, $t = \begin{cases} n.l_n + (K-1), & l_n = K \\ n.K + l_n - 1, & l_n < K \end{cases}$

$$л_i = \prod_{j=n.K, j \neq i}^{t} h_{i,j}^{p_j}$$

4. Compute *VC* over previously computed *VC* up the tree till root

1. If $(HT == 2)$

$$VC_r = \prod_{i=nmodr.K}^{r-1} h_i^{VC_i} \quad \text{where, } n = r$$

$$ri = r$$

2. Else

Initialize: $r_1 = 0$

For $i = 0$ to $HT - 1$

$$r_1 = r_1 + rs_i$$

For $n = r$ to $r_1 - 1$

$$VC_n = \prod_{i=nmodr.K}^{nmodr.K+(K-1)} h_i^{VC_i}$$

$$ri = n$$

5. Compute *л* up the tree over previously computed *VC*

1. If $(HT == 2)$

For $i = q$ to $q + r - 1$

$$л_i = \prod_{j=0, j \neq i}^{r-1} h_{imodq,j}^{VC_j}$$

2. Else

For $n = 0$ to K

For $i = K.n + q$ to $K.n + q + (K-1)$

$$л_i = \prod_{j=n.K, j \neq i}^{n.K+(K-1)} h_{imodq,j}^{VC_j}$$

6. Compute $Root_{VC}$

$$Root_{VC} = VC[ri]$$

Algorithm ***Verkle Tree Proof_Checking*** (p_i, *HT*, *q*, *K*, rs_i, l_i, VC_i, *oldRC*)

1. Determine the packet and the packet's position whose integrity has to be checked
   x→ packet to be checked
   pos→ packet's position value
2. Calculate the verkle proof indexes (*VP*) of *x*
   Initialize: $d\_counter = 0$, $flag = 0$, $VP[0] = pos, counter = 0$ and $u = 0$
   $V \rightarrow [V_0, V_1, V_2, \ldots, V_{HT}]$
   For *i = 1* to *HT*
   While $(j <= q - 1 + u)$
   If $(flag\%K == 0)$
   $counter = counter + 1$
   $flag = 0$
   $flag = flag + 1$
   If $(VP[i-1] == j)$
   $V[i-1] = counter$
   $d\_counter = counter$
   $j = j + 1$
   $VP[i] = q - 1 + d\_counter$
   $u = u + rs[i-1]$
   If $(q\%K! = 0)$
   $counter = counter + 1$

3. Determine the position of the data packets involved in calculating the first *VP* value (VP_0)
   Initialize: $a = 0$
   $n = (V[0] * K) - K$
   If $(V[0] < r)$
   $tt = n + K - 1$
   Else
   $tt = n + l[r-1] - 1$
   Declare an array *p_in*[] of size tt for storing the positions of the data packets involved
   For $z = n$ to tt
   If $(z\ != pos)$
   $p\_in[a] = z$
   $a = a + 1$
4. Calculate *VC* over *x* and the data packet values having the packet positions fetched from step 3
   Initialize: $nV = 1$
   For $i = n$ to tt
   $nV = nV * h_i^{p_i}$
   $nVC[V[0] - 1] = nV$
5. a) Determine the position of the *VC's* involved in next *VP* value
   b) Compute new *VC* over previously computed *nVC* value and the *VC* values having the positions of the determined *VCs* in step 5.a)
   c) Repeat steps 5.a) and 5.b) upto the root (till the end of *VP* array)
   d) Calculate new root commitment (*newRC*) value, which will be the last value in *nVC* array

   Initialize: $newRC = 0, nV = 1$
   For $i = 1$ to $HT - 1$
   $m = 0$
   $e = 0$
   $e = e + rs[i-1] - 1$
   For $b = m$ to e
   If $(b\ != VP[i]\%q)$
   $nVC[b] = VC[b]$
   $nV = nV * h_b^{nVC_b}$
   $nVC[V[i] - 1] = nV$
   $newRC = nVC[(V[i] - 1)]$
6. Compare $newRC$ and *oldRC* value

Comparison of Merkle Tree and Verkle Tree

The performance of the merkle tree and the verkle tree are compared in terms of tree construction time and proof checking time. Tree generation time is the time taken to build the entire tree starting from the first leaf node upto the root. For example, in the case of merkle tree and verkle tree with four packets (as seen in figure. 1 and 3 respectively), the tree generation time is the time taken to build the entire tree starting from the leaf node ($h(p_0)$) and ($p_{0,}$ $_{J0}$) upto the root $h(p_{0123})$ and VC_2 respectively.

Proof checking time is the time taken to check the integrity of a particular packet by evaluating the generated proof of that packet. For example, in the case of merkle tree with four packets (as seen in figure. 1), the merkle proof is ($h(p_2)$, $h(p_{01})$) for the packet p_3. Therefore, proof checking time for p_3 is the time taken to generate this merkle proof, to re-generate the new merkle root value and to compare this new root value to the original root value. Likewise, in the case of verkle tree with four packets (as

seen in figure. 3), the verkle proof is $л_3$, $л_5$ for the packet p_3. Therefore, proof checking time for p_3 is the time taken to generate this verkle proof, to re-generate the new verkle *RC* value and to compare this new *RC* value to the original *RC* value.

The *Verkle Tree Generation* and the *Verkle Tree Proof_Checking* algorithms are implemented in Java, yielding the following results. Figure. 4 shows how the verkle tree generation time changes as the branching factor (*K*) of the tree changes. Here, the number of packets *q* is kept constant ($q = 100$) while varying *K*. From figure. 4, it is clear that, as *K* increases, the verkle tree generation time also increases. In order to reduce the verkle tree generation time, *K* should be kept low. At the same time, *K* also determines the network bandwidth and size of the verkle proof. A low *K* value results in a large verkle proof size, which consumes a lot of network bandwidth. Therefore, in-order to achieve optimal verkle tree generation time, the *K* of the verkle tree should not be too low nor too high. Figure. 5 demonstrates the verkle tree generation time by varying *q* and keeping $K = 32$. From figure. 5, it is clear that, as *q* increases, the verkle tree generation time also increases.

The performance analysis of binary merkle tree and *K*-ary verkle tree ($K = 32$) are done and their comparative results are enumerated in Table 2. When compared to merkle tree, verkle tree takes much longer time to build the tree, but it takes minimal time to check a packet's integrity and has much smaller verkle proof size. According to Table 2, on an average, the tree generation time of a verkle tree is 38 times that of a merkle tree, while the proof checking time is reduced by 98.33%. As a result, once the tree is built, proof checking can be done effectively any number of times. This makes verkle tree ideal for applications where proof checking should be done *N* number of times when compared to tree generation. Therefore, verkle tree is perfect and well-suited for dealing data integrity attacks in OMSN.

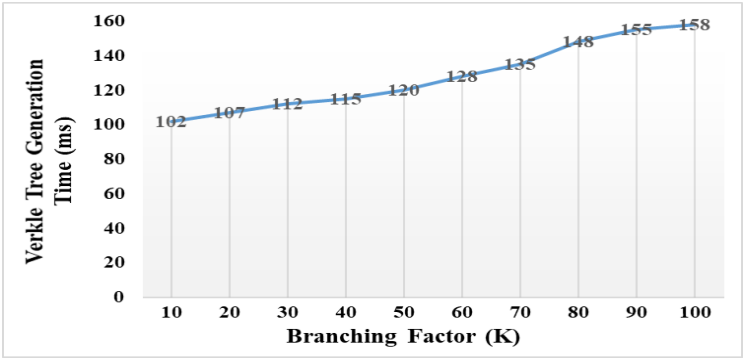


Figure 4: Branching Factor (K) vs Verkle Tree Generation Time

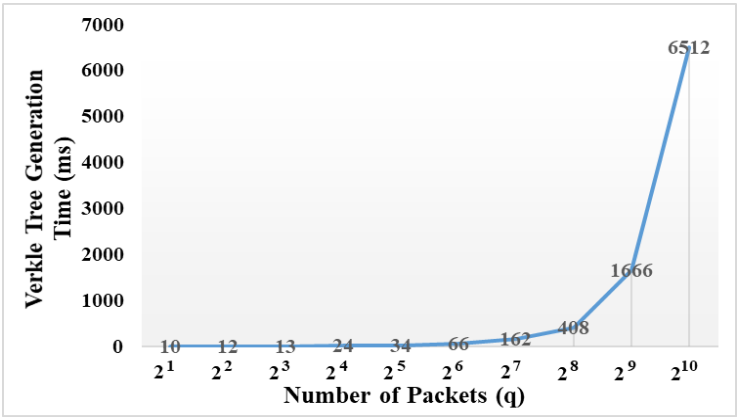


Figure 5: Number of Packets (q) vs Verkle Tree Generation Time

Table 2: Performance Analysis of Merkle Tree and Verkle Tree

Number of Packets (q)	**Tree Generation Time (*ms*)**		**Proof Checking Time (*ms*) (Time taken to verify the integrity of 50th packet)**	
	Merkle Tree	**Verkle Tree**	**Merkle Tree**	**Verkle Tree**
100	14	116	35	0.188
200	17	544	36	0.324
300	19	719	36	0.656
400	22	1124	37	0.907
500	25	1577	38	1.000

VADIA: Verkle Tree on Data Integrity Attacks

One of the most prevalent OMSN attack is the data integrity attack. Verkle tree provides a complete and secure solution for detecting such attacks in OMSN. Malicious nodes once detected are prevented from further routing in the network. In the attack model, only the intermediate nodes are assumed to be involved in malicious activity and each intermediate node, including the destination node, can detect the attack. The following are some of the relevant attacks in OMSN:

Packet dropping attack

Verkle tree is utilized to detect PDA, where source node *S* divides *M* into q number of packets and generates a verkle tree with q leaves. After verkle tree generation, *S* calculates the total number of subsets in verkle tree (*trs*) and total number of membership proofs (*ɲiv*). These values are appended in each packet's header before transmitting the packets to reach destination node *D*. *D* as well as each intermediate nodes on receiving all the data packets, check for PDA by performing the *Packet Drop_Detection* algorithm.

Algorithm ***Packet Drop_Detection*** (*trs, ɲiv*)

1. Calculate original number of packets (q)
   $q = ɲiv - trs + 2$
2. Fetch new number of packets (q')
3. Find malicious node
   If ($q' == q$)
   No packets are dropped
   Else
   Packet drop is detected and mark previous node as malicious
   Add this path to malicious path list
4. Calculate number of packets dropped (*pd*)
   $pd = q - q'$

This algorithm works by computing the total number of packets that is being sent and received by the nodes in the network. Therefore, this algorithm fails if the malicious node performs an attack by preserving the total number of packets even after malicious activity. Following section details two such attacks.

Packet Modification Attack and Pollution Attack

Verkle tree is utilized to detect PMA and PA. It is assumed that the malicious node is capable of modifying only the payload information of the data packet. Initially, source node *S* divides *M* into q number of packets and generates a verkle tree with q leaves. After verkle tree generation, root

commitment value ($Root_{VC}$) is appended in each packet's header before transmitting the packets to reach destination node *D*. *D* as well as each intermediate nodes on receiving all the data packets, check for PMA and PA by performing the *Packet Modify and Pollution_Detection* algorithm. This algorithm detects the modified as well as injected packets. To differentiate between PMA and PA, a parameter namely packet creation time (*pct*) is added to the header of each packet. When a new packet is injected, the *pct* value changes from current packet's *pct* value to new packet's *pct* value. While receiving all the packets, the receiver node also checks for *pct* values. If there is a difference in the *pct* value, then it is a result of PA.

Algorithm ***Packet Modify and Pollution_Detection*** ($Root_{VC}, pct$)

1. For each data packet perform Verkle Tree Proof_Checking algorithm (steps 1-5) and fetch new RC value ($Root_{VC'}$)
   For each p_i
   Call *Verkle Tree Proof_Checking*
   If ($Root_{VC} == Root_{VC'}$) // Find malicious node
   No malicious activity
   Else
   If (p_i*'s pct == pct*)
   PMA, Mark the previous node as malicious
   Add this packet number to the malicious packet list
   Add this path to malicious path list
   Else
   PA, Mark the previous node as malicious
   Add this packet number to the malicious packet list
   Add this path to malicious path list

VADIA Algorithm

The overall working of malicious node detection by the proposed approach is outlined in VADIA algorithm. On receiving all the data packets destination as well as each intermediate node performs this algorithm to check any malicious activity while routing.

Algorithm ***VADIA***

If (transmitted total number of data packets ! = received total number of data packets)

Perform *Packet Drop_Detection* algorithm

Else

Perform *Packet Modify and Pollution_Detection* algorithm

4 Results and Analysis

Simulation Settings

The proposed approach (VADIA) is implemented and the performance is evaluated using ONE simulator (Keränen & Kärkkäinen, 2009). The simulator is configured using Bluetooth interface with transmission range of 50 meters and transmission speed of 250 kBps. The other simulation settings include 50 number of hosts (nodes) having buffer size of 50MB, message Time-To-Live of 300 minutes, message creation interval of 200-500 seconds and message size of 500kB-1MB. These nodes utilize

shortest path map-based movement model and epidemic router for exchanging messages among the nodes in the network.

Performance Metrics

The following are the performance metrics used to evaluate the performance of VADIA algorithm:

- Malicious node detection accuracy: The ratio of the total number of detected malicious nodes to the total number of actual malicious nodes.
- Malicious path detection accuracy: The ratio of the total number of detected malicious paths to the total number of actual malicious paths.
- Dropped/Modified/Faked data detection accuracy: The ratio of the total number of detected dropped/modified/faked packets to the total number of actual dropped/modified/faked packets by the malicious nodes.
- False negative rate: The ratio of the actual malicious nodes detected as legitimate nodes.

The proposed approach is efficient, if the detection accuracies are high and false negative rate is low. When there are two or more malicious nodes in a routing path, only the first malicious node is detected, because after detecting the first malicious node, the path is designated as malicious and further routing is cancelled. The other malicious nodes are detected, only if these nodes re-appear in a different path during network routing. The percentage of malicious nodes is increased from 10% to 50% for different Simulation Time (ST) of 1 hour, 1.5 hours, and 2 hours and the following results are obtained.

a) Malicious Node Detection Accuracy

In figure. 6 (a), (b), and (c) are the results obtained for PDA, PMA, and PA respectively, by using VADIA approach. These figures indicates that as the number of malicious nodes increases, malicious node detection accuracy decreases. This is because, when the number of malicious nodes increases, the number of malicious nodes that appear on a single path increases as well, thus increasing the probability of some of the malicious nodes to go undetected. This leads to low detection accuracy of malicious nodes. At the same time, when ST increases, the chances of such malicious nodes to re-appear in some other path increases, resulting in higher detection accuracy of these malicious nodes. According to the obtained results, the VADIA approach has on an average of 69% detection rate and 47% improvement in malicious node detection accuracy over a period of 2 hours.

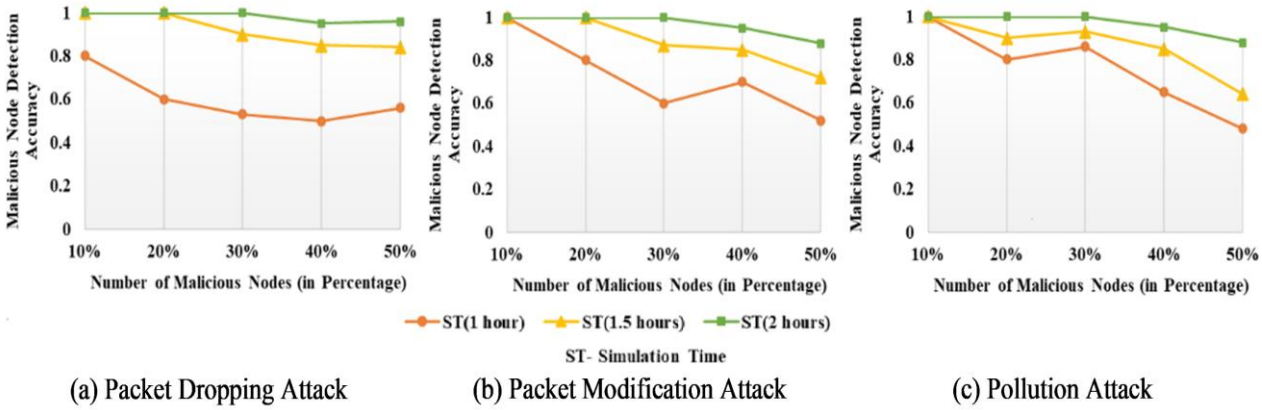


Figure 6: Results Obtained for Malicious Node Detection Accuracy

b) Malicious Path Detection Accuracy

In figure. 7 (a), (b), and (c) are the results obtained for PDA, PMA, and PA respectively, by using VADIA approach. These figures indicates that as the number of malicious nodes increases, malicious path detection accuracy decreases. This is due to the fact that when the number of malicious nodes rises, numerous malicious paths are generated. Therefore, detecting all these paths need a certain amount of time. For this reason, the malicious path detection accuracy is lower when simulation time is less and gets improved as the simulation time increases. As per the findings, the VADIA approach has on an average of 67% detection rate and 84% improvement in malicious path detection accuracy over a period of 2 hours.

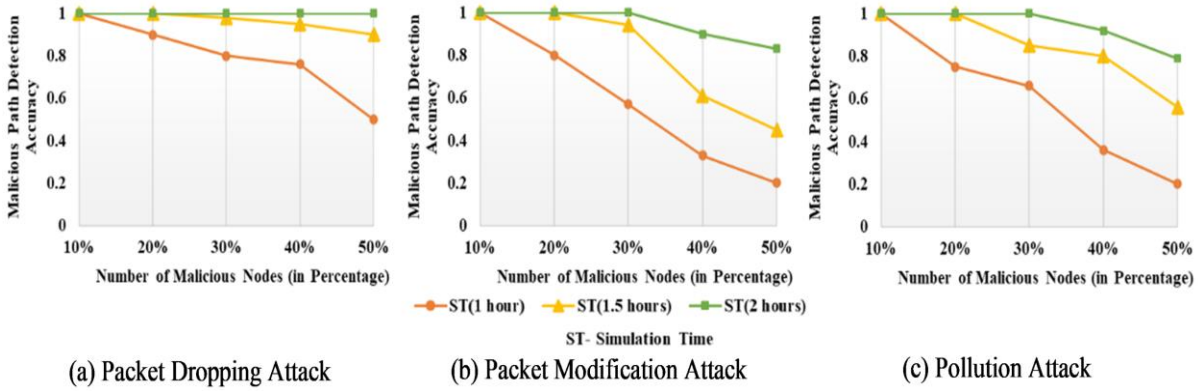


Figure 7: Results Obtained for Malicious Path Detection Accuracy

c) Malicious Activity (Dropped/Modified/Faked Data) Detection Accuracy

In figure. 8 (a), (b), and (c) are the results obtained for PDA, PMA, and PA respectively, by using VADIA approach. These figures portray that as the number of malicious nodes increases, malicious activity detection accuracy decreases. This is because, as number of malicious nodes increases, the malicious activities like dropping, modifying or faking the data increases as well. The malicious nodes are responsible for any malicious activities. Initially the malicious node detection rate will be low which results in low detection of malicious activities, but over a period of time the malicious node detection rate will get improved which in turn leads to higher detection of malicious activities. The results imply that, the VADIA approach has on an average of 62% detection rate and 69% improvement in malicious activity detection accuracy over a period of 2 hours.

d) False Negative Rate

In figure. 9 (a), (b), and (c) are the results obtained for PDA, PMA, and PA respectively, by using VADIA approach. These figures signify that as the number of malicious node increases, false negative rate increases. When there are more malicious nodes, the probability of some of the malicious nodes to encounter in a routing path decrease. This causes low detection of malicious nodes, and increases the probability of such malicious nodes being classified as legitimate nodes. But as ST increases, the probability of these misclassified legitimate nodes being correctly classified as malicious node rises. As per results, the VADIA approach has on an average of 30% misclassification rate and this rate gets reduced by 80% over a period of 2 hours.

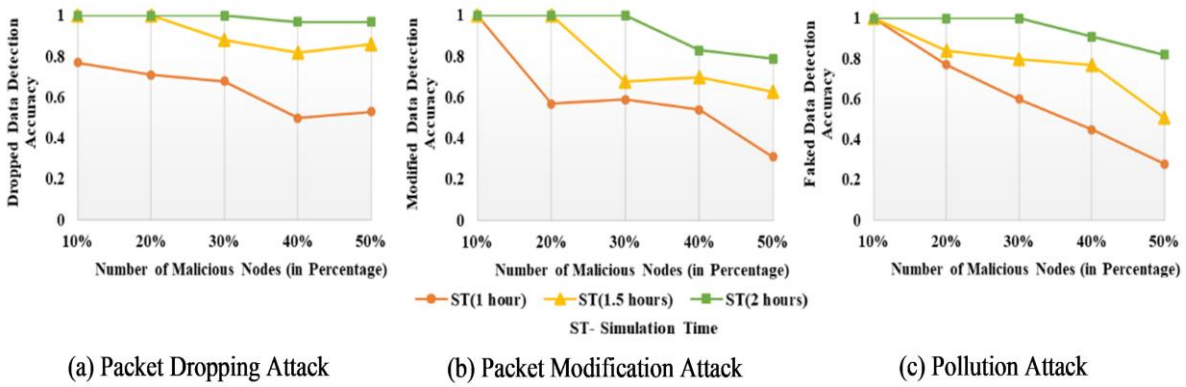


Figure 8: Results Obtained for Malicious Activity Detection Accuracy

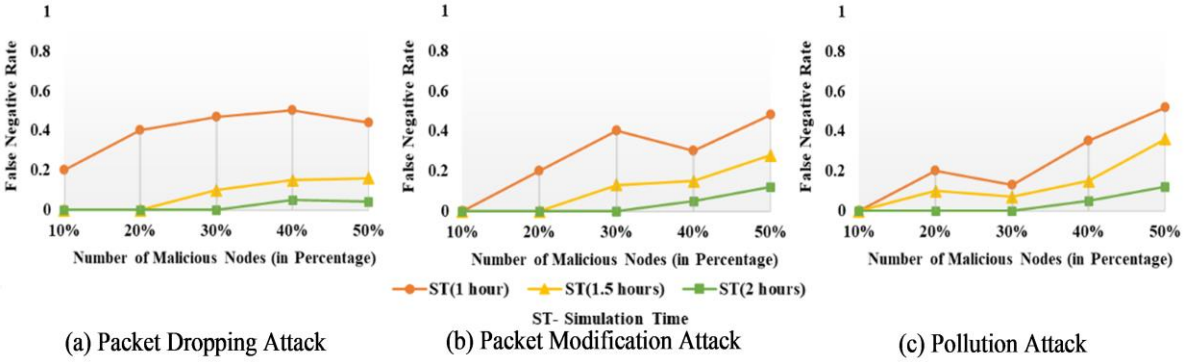


Figure 9: Results Obtained for False Negative Rate

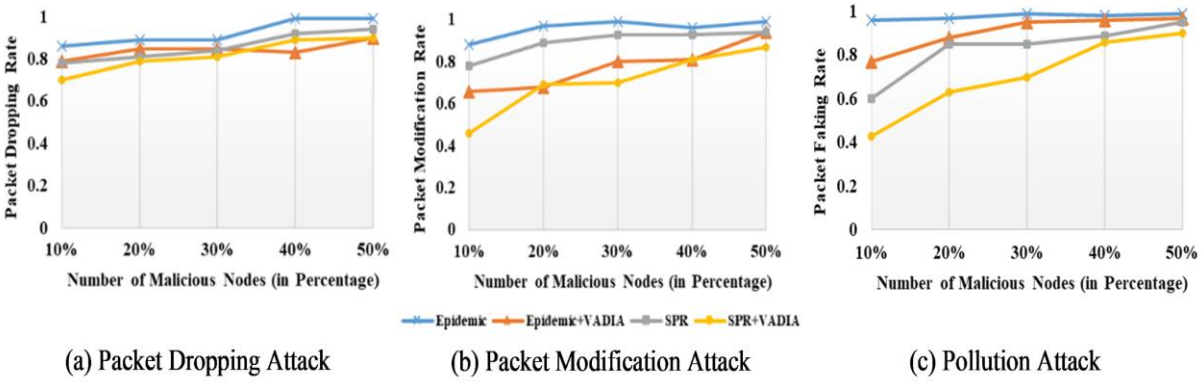
Significance of VADIA Approach

In-order to highlight the significance of the VADIA approach, two routing models namely, Epidemic and Socialized Proficient Routing (SPR) (Lakshmi & Gireesh, 2020) are considered. The former is a benchmark routing model and later is most recent and efficient routing model. The results are extracted with these two routing models by integrating as well as without integrating the VADIA approach. For evaluation three performance metrics are assessed: Packet Dropping Rate (PDR), Packet Modification Rate (PMR) and Packet Faking Rate (PFR). Packet dropping/modification/faking rate is the ratio of total number of dropped/modified/faked packets to the total number of packets in the network. These performance metrics should be low for a good model.

Figure. 10 and 11 are the results obtained by increasing the number of malicious nodes over a period of 1 hour and increasing ST for 25% of malicious nodes respectively. In Figure. 10 and 11, (a), (b), and (c) are the graphs for PDA, PMA, and PA assessing PDR, PMR, and PFR respectively. It is seen in figure. 10 that, as the number of malicious nodes increases, the packet dropping/modification/faking rates increases. This is due to the fact that, as the number of malicious nodes increases, the malicious activities also increase. The VADIA approach controls these malicious activities by detecting and preventing the malicious nodes during routing. For the routing models without VADIA approach the packet dropping/Modification/Faking rates are higher when compared to routing models with the VADIA approach. Without VADIA approach almost 95.33% and 86% of total data is compromised in Epidemic and SPR model respectively. But with VADIA approach, only 84.26% and 74.27% of total data is compromised in Epidemic and SPR model respectively. On an average with the utilization of

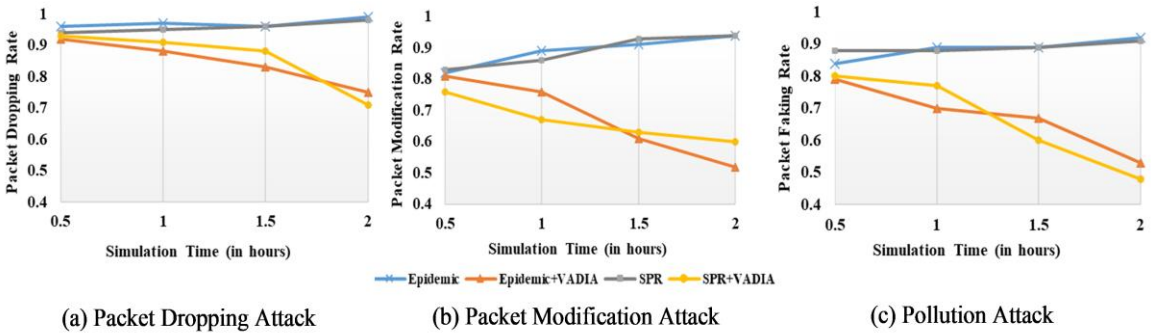
VADIA approach the malicious activity rates are reduced by 8.6% in Epidemic model and 14% in SPR model by increasing the malicious nodes upto 50% of total nodes in the network.

From Figure. 11 it is conveyed that, as ST increases, in the routing models without VADIA approach the packet dropping/modification/faking rates increases, since there is no control over the malicious nodes and their activities. But, for the routing models with VADIA approach the packet dropping/modification/faking rates decreases, as the malicious nodes are detected and prevented from data exchange during the course of time. Without VADIA approach almost 91.5% and 91.25% of total data is compromised in Epidemic and SPR model respectively. But with VADIA approach, only 73.08% and 72.83% of total data is compromised in Epidemic and SPR model respectively. On an average with the utilization of VADIA approach the malicious activity rates are reduced by 19.92% in Epidemic model and 20.13% in SPR model over a period of 2 hours.



(a) Packet Dropping Attack (b) Packet Modification Attack (c) Pollution Attack

Figure 10: Results Obtained by Increasing the Number of Malicious Nodes



(a) Packet Dropping Attack (b) Packet Modification Attack (c) Pollution Attack

Figure 11: Results Obtained by Increasing Simulation Time

5 Conclusion and Future Work

In this paper two tree-based data structures namely, merkle tree and verkle tree are explored. These two data structures are compared based on tree generation time and proof checking time for the purpose of dealing data integrity attacks in OMSN. From the analysis, it was evident that verkle tree had smaller proof size and performed better with shorter proof checking time (98.33% reduction), when compared with merkle tree. Due to these characteristics, verkle tree is significantly more appropriate for applications that requires frequent proof checking than tree generation. One such application is malicious node detection in OMSN.

Three major attacks such as PDA, PMA and PFA are investigated in this work. A novel technique known as VADIA is utilized for detection and prevention of the above attacks in OMSN. The results are analyzed based on the performance metrics like malicious node, malicious path, and malicious activity detection accuracy as well as false negative rate. The results demonstrated high malicious node, path as well as activity detection accuracy and low false negative rate. In accordance with the findings, VADIA approach had 69%, 67%, 62% malicious node, path, and activity detection rate respectively and 30% false negative rate. The values are improved by 47%, 84%, 69% and 80% over a period of 2 hours.

Finally, the significance of the proposed approach (VADIA) is indicated by incorporating it in two routing models (Epidemic and SPR). The PDR, PMR and PFR is compared for these two routing models with and without VADIA approach. The results obtained verified that the routing models with VADIA approach had lower PDR, PMR and PFR, when compared to the routing models without VADIA approach. With the utilization of VADIA approach over 50% of malicious nodes, on an average these rates are reduced by 8.6% and 14.1% for Epidemic and SPR model respectively. Also, with the utilization of VADIA approach over a period of 2 hours, on an average these rates are reduced by 19.92% and 20.13% for Epidemic and SPR model respectively.

Future Work: Besides the three significant data integrity attacks evaluated in this paper, future work can be extended for additional data integrity attacks like collusion attack (Alajeely et al., 2017), and catabolism attack (Alajeely, Doss & Mak-Hau 2015). Also, other types of attacks like packet flooding attacks/ Denial-Of-Service attacks (replay attacks, black hole attack (Barai & Bhaumik, 2021), jamming attack (Singh, Woungang, Dhurandher, & Khalid, 2022), worm hole attack (Dhurandher, Singh, Nicopolitidis, Kumar & Gupta, 2022)), sybil attacks and selfish attacks can be addressed in future work. Furthermore, an efficient routing model that incorporates an attack model that detects and prevents all these attacks is demanding and challenging.

References

[1] Ahmad, A.A., Doss, R., Alajeely, M., & Rubeaai, S.F.A. (2018). Trust strategy implementation in OppNets. *Computing*, *100*, 151-181.

[2] Alajeely, M., & Doss, R. (2014). Defense against packet dropping attacks in opportunistic networks. *In IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 1608-1613.

[3] Alajeely, M., & Doss, R. (2015a). Malicious node traceback in opportunistic networks using merkle trees. *In IEEE International Conference on Data Science and Data Intensive Systems*, 147-152.

[4] Alajeely, M., & Doss, R. (2015b). Malicious node detection in OppNets using hash chain technique. *In IEEE 4th International Conference on Computer Science and Network Technology (ICCSNT)*, *1*, 925-930.

[5] Alajeely, M., & Doss, R. (2016). Establishing trust relationships in OppNets using Merkle trees. *In IEEE 8th International Conference on Communication Systems and Networks (COMSNETS)*, 1-6.

[6] Alajeely, M., & Doss, R. (2016). Reputation based malicious node detection in OppNets. *In IEEE 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 1-6.

[7] Alajeely, M., Ahmad, A.A., Doss, R., & Mak-Hau, V. (2015). An Efficient Detection Mechanism Against Packet Faking Attack in Opportunistic Networks. *In Future Network Systems and Security: First International Conference, FNSS, Paris, France, Proceedings 1*, 84-100.

[8] Alajeely, M., Doss, R., & Mak-Hau, V. (2015). Catabolism attack and anabolism defense: A novel attack and traceback mechanism in opportunistic networks. *Computer Communications*, *71*, 111-118.

[9] Alajeely, M., Doss, R., & Mak-Hau, V. (2017). Defense against packet collusion attacks in opportunistic networks. *Computers & Security*, *65*, 269-282.

[10] Altaweel, A., Stoleru, R., Gu, G., Maity, A.K., & Bhunia, S. (2022). On detecting route hijacking attack in opportunistic mobile networks. *IEEE Transactions on Dependable and Secure Computing, 20*(3), 2516 - 2532.

[11] Barai, S., & Bhaumik, P. (2021). Detection and Mitigation of Blackhole Attack Effect in Opportunistic Networks. *In IEEE 19th OITS International Conference on Information Technology (OCIT)*, 155-160.

[12] Catalano, D., & Fiore, D. (2013). Vector commitments and their applications. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan. Proceedings.* Springer Berlin Heidelberg, *16*, 55-72.

[13] Dhurandher, S.K., Singh, J., Nicopolitidis, P., Kumar, R., & Gupta, G. (2022). A blockchain-based secure routing protocol for opportunistic networks. *Journal of Ambient Intelligence and Humanized Computing*, 1-13.

[14] Doss, R., Alajeely, M., & Al Rubeaai, S.F. (2018). Packet integrity defense mechanism in OppNets. *Computers & Security*, *74*, 71-93.

[15] Fall, K. (2003). A delay-tolerant network architecture for challenged internets. *In Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications*, 27-34.

[16] Gupta, M. (2019). Cyclic redundancy checks based data authentication in opportunistic networks. *In 2nd International Conference on Wireless Intelligent and Distributed Environment for Communication: WIDECOM*. Springer International Publishing, 17-26.

[17] Kaur, M., & Mahajan, M. (2013). Using encryption algorithms to enhance the data security in cloud computing. *International Journal of Communication and Computer Technologies (IJCCTS)*, *1*(2), 130-133.

[18] Keränen, A., Ott, J., & Kärkkäinen, T. (2009). The ONE simulator for DTN protocol evaluation. *In Proceedings of the 2nd international conference on simulation tools and techniques*, 1-10.

[19] Kuszmaul, J. (2019). Verkle trees. *Verkle Trees*, 1-12

[20] Lakshmi, V.R.V., & Gireesh, K.T. (2019). Opportunistic mobile social networks: architecture, privacy, security issues and future directions. *International Journal of Electrical and Computer Engineering*, *9*(2), 1145-1152.

[21] Lakshmi, V.R.V., & Kumar, T.G. (2017). Mobile Social Networks: Architecture, Privacy, Security Issues and Solutions. *J. Commun.*, *12*(9), 524-531.

[22] Lakshmi, V.R.V., & Thonnuthodi, G.K. (2020). Socialized proficient routing in opportunistic mobile network using machine learning techniques. *International Journal of Intelligent Engineering and Systems*, *13*(4), 434-445.

[23] Merkle, R.C. (1980). Protocols for public key cryptosystems. *In IEEE symposium on security and privacy*, 122-122.

[24] Pieters, W. (2011). Representing humans in system security models: An actor-network approach. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JOWUA), 2*(1), 75-92.

[25] Rashidibajgan, S. (2017). A Solution for Prevention of Selective Dropping and Selfish Attacks in Opportunistic Networks. *In International Conference on Internet of Things, Big Data and Security*, SCITEPRESS, *2*, 30-35.

[26] Rashidibajgan, S., Hupperich, T., Doss, R., & Förster, A. (2021). Secure and privacy-preserving structure in opportunistic networks. *Computers & Security*, *104*, 102208. https://doi.org/10.1016/j.cose.2021.102208

[27] Singh, J., Woungang, I., Dhurandher, S. K., & Khalid, K. (2022). A jamming attack detection technique for opportunistic networks. *Internet of Things*, *17*, 100464. https://doi.org/10.1016/j.iot.2021.100464
[28] Sonya, A., & Kavitha, G. (2022). A Data Integrity and Security Approach for Health Care Data in Cloud Environment. *Journal of Internet Services and Information Security*, *12*(4), 246-256.
[29] Wang, K., & Sakai, K. (2020). Randomized authentication using IBE for opportunistic networks. *In 49th International Conference on Parallel Processing-ICPP: Workshops*, 1-7.

Authors Biography

Vimitha R Vidhya Lakshmi is currently pursuing her Ph.D. degree with TIFAC-CORE in cyber security, Amrita University, Coimbatore, India. She received her B. Tech. degree in Information Technology and her M. Tech. degree in Computer and Information Science from the Cochin University of Science and Technology (CUSAT), India, in 2012 and 2015 respectively. Her research interests include mobile computing, information security and wireless communication.

Dr.T. Gireesh Kumar holds his Ph.D. degree in Computer Science and Engineering from Anna University, India, in 2011. He received his M. Tech. degree in Computer Science and Engineering from the Cochin University of Science and Technology (CUSAT), India, in 2003. He is currently working as Associate professor in Computer Science and Engineering department at Amrita University, India. His research interests include wireless communication, machine learning, algorithm and artificial intelligence.