DOI 10.5162/iCCC2026/P33

Securing Custom IEEE 802.11 Based Mesh Networks on ESP32 IoT Modules

Pavlo Mykytyn*, Randolf Rotta*, BTU Cottbus-Senftenberg, Cottbus, Germany
mykytyn@b-tu.de, rottaran@b-tu.de
*The authors contributed equally to this research

Abstract

ESP32-based wireless modules have become a popular and low-cost platform for prototyping IoT and distributed sensing applications. The integrated radios for IEEE 802.11 Wi-Fi, BLE, and IEEE 802.15.4 communication offer a flexible foundation for ad-hoc networking. Existing mesh network implementations for the ESP32 that support encryption for security are limited to an inflexible tree topology of access points. Meanwhile, mesh protocols like B.A.T.M.A.N. for ESP32, which offer fast route repair and throughput optimized route selection, lack implementation of security mechanisms. To close this gap, we first implemented and evaluated a software-based encryption scheme utilizing ECC and AES-GCM for the ESP32 B.A.T.M.A.N mesh. However, the software implementation revealed significant performance limitations, motivating the reverse engineering of the ESP32 radio driver to enable and implement on-the-fly hardware encryption/decryption. This approach makes it possible to secure custom mesh protocols without impairing throughput and processor load.

1 Introduction

Ad-hoc mesh networks provide communication where existing infrastructure is not easily available, does not match the application's communication patterns, or an infrastructure independent solution is desired. Self-organizing mesh protocols can be rapidly deployed and can detect and repair broken routes quickly. Some applications demand mobile deployment and cannot rely on pre-existing communication infrastructure. Examples include temporary field operations [1], autonomous robotic systems [2] or low-power IoT networks in remote areas [3], [4]. Some applications benefit from direct communication across peer devices. For example, cooperative collision avoidance and path planning in mobile swarms relies on periodic position broadcasts and negotiation between nearby vehicles.

The practical usefulness of such self-organizing mesh networks highly depends on their security. Their distributed, decentralized, and sometimes intentionally open nature increases the challenges for ensuring confidentiality, authenticity and integrity of messages as well as detecting and preventing attacks on the link and networking layer, as stated in [5]. Without well-designed security, mesh networks can become a serious safety risk, as described in [6]. Many routing protocols, including the B.A.T.M.A.N (Better Approach To Mobile Ad hoc Networking) [7] mesh protocol, do not incorporate encryption at the network layer, delegating it instead to the application layer. The commonly used Linux kernel implementations can be combined with 802.11s security mechanisms, however, this is not yet available for constrained hardware like the ESP32-S3, C3, or C6 from Espressif Systems. Moreover, Espressif's SDK does support 802.11s, nor does it expose the required mesh peering and key management APIs.

In this work, we extend the implementation of a B.A.T.M.A.N routing protocol with a software-based and hardware-based implementations of AES encryption mechanisms to improve the network security.

2 Background and Related Work

The routing protocol is responsible for discovering and selecting routes between source and destination nodes as well as forwarding packets along these routes. A flooding method is often used for discovering routes. Mesh routing outperforms tree-based routing in high-mobility networks, offering alternative paths for packet transmission from source to destination. The route discovery in these protocols can be on-demand like in the AODV protocol [8], proactive like in the OLSR [9], BABEL [10], B.A.T.M.A.N, or hybrid such as HWMP [11]. Proactive protocols keep routes to every node updated at all times, even if those routes are never used. On-demand protocols only look for a route when it's needed or when one fails.

To the best of our knowledge, there is no open mesh implementation for low-power microcontrollers based on IEEE 802.11 transceivers. To close this gap, we implemented the B.A.T.M.A.N protocol and the required software stack on the Espressif ESP32 platform [4].

2.1 Attack Vectors on B.A.T.M.A.N Meshes

B.A.T.M.A.N-adv extends the original B.A.T.M.A.N protocol by shifting routing from the network layer to the data link layer and forwards frames based on MAC addresses instead of IP addresses. This design simplifies multi-hop communication, making it better suited for lightweight, dynamic mesh deployments.

However, even the advanced version of B.A.T.M.A.N does not include any built-in encryption, authentication, or integrity checks. All control messages, such as originator messages and topology updates are sent in plaintext. An eavesdropper can, therefore, learn every node's address and link metrics, effectively revealing the mesh topology. Because there is no cryptographic validation, any node can inject or spoof routing packets. In practice this exposes the mesh to:

Eavesdropping attacks, where a passive listener can learn the entire network topology and eavesdrop on the exchanged messages.
Flooding attacks, where malicious nodes can flood the network with fake originator messages and malformed frames to overwhelm the legitimate routing updates.
Blackhole attacks, where an attacker falsely announces the best route to a destination, claiming to be the next hop, and drawing traffic into a black hole.
Wormhole attacks, where legitimate packets are relayed over large distances in order to create a false sense of proximity and attract more traffic for eavesdropping.

2.2 Wi-Fi WPA3 and 802.11s Security

WPA3-Personal uses Simultaneous Authentication of Equals (SAE), a mechanism that was originally developed for 802.11s wireless mesh networks [13]. Based on SAE, a pairwise secured channel is established and then used to establish a Group Master Key (GMK) and Pairwise Master Key (PMK). A Group Temporal Key (GTK) is derived from the GMK and is used to encrypt broadcast frames. A Pairwise Transient Key (PTK) is derived from the PMK and parts of it are used to derive the Temporal Key (TK) that is used to encrypt unicast frames. On Linux-based single-board computers, 802.11s based peer authentication and encryption can be combined with custom mesh protocols such as BATMAN.
Although the ESP32 firmware does not implement 802.11s routing, the WPA3 personal and WPA3 security is implemented and the hardware supports the respective symmetric ciphers. The encryption capabilities of the radio hardware were first described by [14] based on reverse engineering of the Wi-Fi SoftMAC driver. Espressif offers a proprietary, closed-source mesh called ESP-MESH, based on the legacy WPA2 standard and tree-based routing [15]. Mesh security also depends on managing group membership. WPA3-Personal lacks any standard method to revoke keys or remove peers. Once connected, a node retains the key indefinitely. This is a major issue for any mesh with shifting membership. WPA3-Enterprise addresses this through a central authentication service based on RADIUS. However, forwarding the distributed association requests via a resource-constrained mesh network to this central authenticator is susceptible to denial-of-service attacks.

3 Authenticated Key Agreement and Software-based Encryption

To secure the communication in a mesh network, we first implemented and tested a mechanism that uses Elliptic Curve Cryptography (ECC) combined with a software implementation of symmetric encryption using Advanced Encryption Standard (AES) in Galois-Counter Mode (GCM). We have already partially discussed this approach in our earlier works [3], [16]. The key agreement process is conducted pairwise to establish a session key. We use Elliptic Curve Diffie-Hellman (ECDH) key exchange between each node and the base station. However, ECDH alone does not guarantee authenticity and does not protect against Man-in-the-Middle attacks. Therefore, to authenticate the key agreement process, both the base station and each node are provided with their own Elliptic Curve Digital Signature Algorithm (ECDSA) keys, stored in the non-volatile storage (NVS). Once the pairwise ECDH key agreement is initiated, the base station signs its public ECDH key with its private ECDSA key and distributes it to each node in the mesh. Upon receipt, each party verifies the authenticity of the ECDH public keys and then uses them to compute a shared secret and generate a session key.
Once a session key has been established, it is used to generate further pairwise keys using a Key Derivation Function (KDF) and symmetrically encrypt the communication and ensure forward secrecy. The multi-hop mesh communication is encrypted using software-implementation of the 256-bit AES-GCM, based on the open-source Mbed-TLS library [17] with hardware-specific optimizations for the ESP32. It provides authenticity and integrity by performing Galois field multiplication of the ciphertext with a hash of the plaintext and associated data. The throughput and performance testing of this implementation are presented in the next section.

3.1 Throughput Benchmarks

We used ESP32-S3 modules with 16 MB of flash memory, 8 MB of external SPI-RAM, and an on-board PCB antenna. They were programmed using ESP-IDF v5.1.4 with the CPU frequency set to 240 MHz, flash memory frequency to 120 MHz, and SPI-RAM frequency also to 120 MHz, ensuring optimal performance for our experiments.
Seven nodes were arranged along a hallway, spaced 10 meters apart from each other, to create a mesh network as presented on the Figure 1. To enforce multi-hop communication, the transmit power was reduced to 8 dBm. The longest communication route spanned 4 hops. The throughput was measured by continuously sending 1400-byte blocks for 5 seconds, with the number of received bytes recorded every 250 ms on the receiving node, resulting in 20 throughput samples for each connection type.

The first configuration ran both encryption/decryption with AES-GCM and a 256-bit key and routing tasks on core 0. It is marked with blue color in the Figure 2. In the second setup, encryption/decryption was done on core 0, while the routing task ran on core 1. It is marked with orange color in the Figure 2. As a baseline, the third setup used no encryption and is marked with green color on the Figure 2.

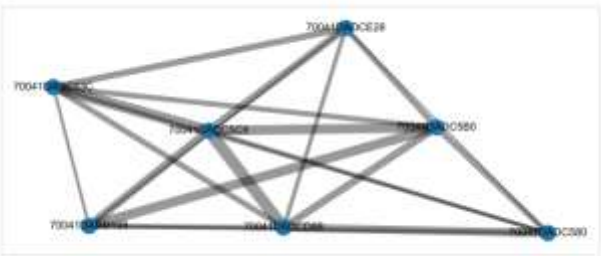

Figure 1 Node layout configuration of the mesh network

DOI 10.5162/iCCC2026/P33

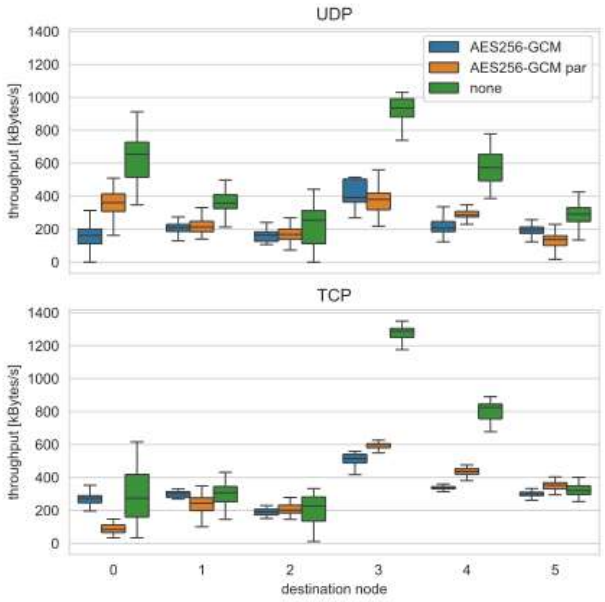

Figure 2 Averaged throughput of up to 5 hops, over UDP and TCP connections.

The software-based encryption shows a significant reduction of throughput on high-throughput single-hop routes: Without encryption, the highest measured throughput between a node pair was around 1300 kBytes/s and with encryption, the throughput between the same pair dropped to around 400 kBytes/s. On longer multi-hop routes, the measured throughput did not exceed 400 kBytes/s without encryption. Fortunately, the throughput with encryption did not drop significantly. This can be explained due to the additional processing overhead caused by encryption and decryption mostly overlapping with delays caused by the collisions between incoming and forwarded frames.

4 Hardware-Based Encryption in ESP32 Wi-Fi Radio

Newer versions of the ESP32 SoC, such as S3 and C6 support WPA3-Enterprise and, hence, hardware-based encryption for AES-GCMP with 256-bit keys. However, because of the closed source radio driver, this was not accessible to custom mesh network implementations. This section summarizes insights from partial reverse-engineering and proposes a programming interface to simplify the integration into custom network stacks.

4.1 Encryption Key Slot Architecture

The 802.11 radio integrated in the ESP32 SoC can perform encryption and decryption of frames on the fly. For this, the symmetric temporal key needs to be placed into one of the 25 available hardware key slots. Each key slot also contains the configuration of the cipher algorithm, a MAC address, the 2-bit WPA key ID to select the right key for incoming frames, the hardware interface (STA/AP) the key applies to, and a few configuration flags to match, for example, broadcast frames. The WPA key ID is used for periodic key rollover.

A few internal undocumented functions can be used to install own keys. However, they assume, that group temporal keys for broadcast frames are always placed in the first four slots. Higher layers of the driver manage the slot assignment and periodic key rollover. The station interface needs just four slots: two for the group key and two for the pairwise key towards the access point. The access point interface needs two slots for the group temporal key and two slots for each associated station. Thus, at most 9 stations can connect to the access point.

The frame format follows the IEEE 802.11 standard. If the Protected Frame flag is set in the header, an 8-byte WPA header follows, which contains the unique packet number and WPA key ID. Then follows the payload and, finally, a 16 bytes Message Integrity Check (MIC) code. The protection flag, WPA header, and MIC area have to be inserted manually before transmission. The radio will then compute and verify the MIC transparently.

The radio manages multiple pending transmission and the hardware key slot number needs to be configured in the respective transmit slot in order to apply the desired encryption key. The low-level transmit functions read this key slot number from the frame's metadata. The documented transmit function for custom frames does not configure this part of the metadata, but it is also not changed later.

Decryption of received frames is done transparently. For unicast frames, the transmitter address from the frame header and the WPA key ID are used to find a matching entry in the hardware key slots. For broadcast frames, the first key slot that matches the interface (STA/AP) and WPA key ID is used while the frame's BSSID address is ignored. When the cryptographic verification fails, the frame is still acknowledged to the transmitter but discarded. If no hardware key slot matches, the frame is not decrypted but still reported via the monitor mode callback.

4.2 Proposed Programming Interface for Custom Mesh Networks

The programming interface should make it possible to manage a few reserved key slots for associated peers and broadcasts, select the right key for outgoing frames, and identify received frames that were not decrypted because no key slot matched. A low-level approach would be to provide direct access to the reserved key slots in order to directly install and configure keys. In this case, the transmit function for custom frames needs an additional argument to select the encryption key slot. We followed this approach for our own experiments.

A more comfortable interface would allow to manage a list of associated peers based on their MAC address. Then, the assignment to key slots would be handled internally and the transmit function transparently looks up the correct key slot for each outgoing frame. An example for such interface is the peer management in Espressif's ESP-NOW protocol, which allows to configure a key for each associated peer.

Although more comfortable, this interface has a couple of drawbacks. Most importantly, different strategies exist for the key rollover and respective use of multiple key slots. A

high-level management would mandate a specific strategy. For example, the mesh protocol can use clock synchronization to coordinate key rollovers such that a single slot per peer is sufficient. Secondly, the mesh routing layer selects the next-hop peer for forwarding and, hence, already knows the key slot such that the internal search would not be necessary.

Finally, the 802.11s standard uses a different Group Temporal Key on each mesh node for broadcast frames. Thus, it would be necessary to also manage this key for each peer. However, according to our reverse engineering efforts, the hardware does not support multiple broadcast keys because of the key slot selection process for received broadcast frames. In addition, even with a workaround, this needs a second installed broadcast key for each peer and, thus, will reduce the maximum number of peers from 20 to 10, which would severely restrict the mesh protocol. Instead, we propose to use a mesh-wide group temporal key for broadcasts in combination with time synchronized key rollover.

5 Conclusions

ESP32 modules have become a popular low-cost platform for IoT applications. The introduction of 802.11-based self-organizing mesh networks enables new applications. These applications require confidentiality, integrity, and availability in the wireless communication. This paper shows that software and hardware encryption can be integrated into custom mesh implementations. However, software-based encryption causes additional processing overhead that significantly reduces the peak throughput. This can be overcome by making use of the radio's built-in hardware cryptography.

References

[1] A. G. Q. Aquino, A. H. Ballado, and A. V. Bautista, "Implementing a Wireless Sensor Network with Multiple Arduino-Based Farming Multi-Sensor Tool to Monitor a Small Farm Area Using ESP32 Microcontroller Board," in *2021 IEEE 13th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, Nov. 2021, pp. 1–6. doi: 10.1109/HNICEM54116.2021.9731989.

[2] "Heterogeneous Ground-Air Autonomous Vehicle Networking in Austere Environments: Practical Implementation of a Mesh Network in the DARPA Subterranean Challenge." Accessed: Nov. 10, 2025. [Online]. Available: https://www.computer.org/csdl/proceedings-article/dcoss/2022/951200a261/1GBSOhnaUvu

[3] R. Rotta and P. Mykytyn, *Secure Multi-hop Telemetry Broadcasts for UAV Swarm Communication*. 2024. doi: 10.26127/BTUOpen-6637.

[4] R. Rotta, J. Schulz, B. Naumann, N. S. Chatharajupalli, J. Nolte, and M. Werner, "B.A.T.M.A.N. Mesh Networking on ESP32's 802.11," in *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, Oct. 2024, pp. 1–7. doi: 10.1109/LCN60385.2024.10639698.

[5] M. A. Lopez, M. Baddeley, W. T. Lunardi, A. Pandey, and J.-P. Giacalone, "Towards Secure Wireless Mesh Networks for UAV Swarm Connectivity: Current Threats, Research, and Opportunities," in *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, July 2021, pp. 319–326. doi: 10.1109/DCOSS52077.2021.00059.

[6] M. R. Albrecht, J. Blasco, R. B. Jensen, and L. Mareková, "Mesh Messaging in Large-scale Protests: Breaking Bridgefy," 2021, 2021/214. Accessed: Nov. 10, 2025. [Online]. Available: https://eprint.iacr.org/2021/214

[7] "Doc-overview - batman-adv - Open Mesh." Accessed: Nov. 10, 2025. [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki

[8] S. R. Das, C. E. Perkins, and E. M. Belding-Royer, "Ad hoc On-Demand Distance Vector (AODV) Routing," Internet Engineering Task Force, Request for Comments RFC 3561, July 2003. doi: 10.17487/RFC3561.

[9] T. H. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," Internet Engineering Task Force, Request for Comments RFC 3626, Oct. 2003. doi: 10.17487/RFC3626.

[10] J. Chroboczek and D. Schinazi, "The Babel Routing Protocol," Internet Engineering Task Force, Request for Comments RFC 8966, Jan. 2021. doi: 10.17487/RFC8966.

[11] M. A. Ng and K.-L. A. Yau, "An energy-efficient Hybrid Wireless Mesh Protocol (HWMP) for IEEE 802.11s mesh networks," in *2013 IEEE International Conference on Control System, Computing and Engineering*, Nov. 2013, pp. 17–21. doi: 10.1109/ICCSCE.2013.6719925.

[12] "batman-adv — The Linux Kernel documentation." Accessed: Nov. 11, 2025. [Online]. Available: https://www.kernel.org/doc/html/v4.15/networking/batman-adv.html

[13] "IEEE 802.11s — Linux Wireless documentation." Accessed: Nov. 14, 2025. [Online]. Available: https://wireless.docs.kernel.org/en/latest/en/developers/documentation/ieee80211/802.11s.html

[14] J. Devreker, "Reverse engineering ESP32 Wi-Fi driver: the road ahead," ESP32 open MAC. Accessed: Nov. 14, 2025. [Online]. Available: https://esp32-open-mac.be/posts/0005-the-road-ahead/

[15] "ESP-WIFI-MESH - ESP32 - — ESP-IDF Programming Guide v5.5.1 documentation." Accessed: Nov. 14, 2025. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/esp-wifi-mesh.html

[16] P. Mykytyn, M. Brzozowski, Z. Dyka, and P. Langendoerfer, *Towards Secure and Reliable Heterogeneous Real-time Telemetry Communication in Autonomous UAV Swarms*. 2024, p. 165. doi: 10.5162/iCCC2024/P15.

[17] "Mbed TLS · GitHub." Accessed: Nov. 12, 2025. [Online]. Available: https://github.com/Mbed-TLS