Utilizing the Messaging Layer Security Protocol in a Lossy Communications Aerial Swarm

Elyssa Dietz
Naval Postgraduate School
elyssa.dietz@jpl.nasa.gov[1]

Duane Davis
Naval Postgraduate School
dtdavi1@nps.edu

Britta Hale
Naval Postgraduate School
britta.hale@nps.edu

Abstract

Recent advancements in unmanned aerial vehicle (UAV) capabilities have led to increasing research into swarming systems. Unfortunately, efforts to date have not resulted in viable secure communications frameworks, and the limited processing power and constrained networking environments that characterize these systems preclude the use of many existing secure group communications protocols. The Messaging Layer Security (MLS) protocol, currently under development at the Internet Engineering Task Force (IETF), offers some attractive properties for these types of systems. This work looks at integrating MLS into the Advanced Robotic Systems Engineering Laboratory (ARSENL) UAV swarm system as a means of assessing its efficacy. Implementation test results are presented both for experiments conducted in a simulation environment and with physical UAVs.

Keywords: Uncrewed systems, UxS, messaging layer security, network security, UAV, swarm security

1. Introduction

Recent advances in unmanned aerial vehicle (UAV) capabilities have made multi-vehicle and swarming systems potentially applicable to a wide array of military and civilian applications. Tactical utilization of UAV swarms will depend, however, on secure communications. Unfortunately, individual swarming platforms have limited processing power, and swarm systems often utilize unreliable and bandwidth-limited communications frameworks that limit the ability to implement security features. This calls the ability of these systems to meet security requirements associated with classified or sensitive missions into question.

Many current methods of securing communication for groups of devices are unlikely to be applicable to existing or envisioned swarm systems. Recent research in secure group communications, however, indicates that the Messaging Layer Security (MLS) protocol [1] can provide an attractive option with characteristics that seem particularly suited to these types of systems. This protocol provides a computationally efficient way to implement asynchronous secure group key management, but experimentation in realistic systems is required to assess the protocol's functionality in these computational- and communications-limited environments.

The MLS protocol provides a number of capabilities that are particularly relevant to multi-UAV systems including the ability to dynamically add and remove members without sacrificing continuous secure communications among active members of the group. The MLS protocol also makes it possible to forcibly remove UAVs that have been hijacked, compromised, or are malfunctioning. In these situations, the protocol provides the group with a means of updating the communication keys to exclude the compromised or malfunctioning UAV.

This paper describes the implementation of MLS on the Naval Postgraduate School (NPS) Advanced Robotic Systems Engineering Laboratory (ARSENL) UAV swarm system using open-source C++ code maintained by Cisco Systems [2]. In particular, the MLS group operations for key update, member addition, and member removal are implemented and tested. The research includes an analysis of the effect of MLS protocol utilization on ARSENL swarm performance. Specific contributions of this work include the following:

- a proof of concept MLS implementation on a

[1]This work was done as a private venture and not in the author's capacity as an employee of the Jet Propulsion Laboratory, California Institute of Technology. Sponsorship to present this work at the Hawaiʻi International Conference on System Studies was provided by the Jet Propulsion Laboratory, California Institute of Technology, under contract to NASA, the National Aeronautics and Space Administration.

[2]Research reported in this publication was supported by ONR under award number N0001422WX01882 and NSF under agreement number 1565443.

real-world UAV swarm system,

- characterization of MLS performance in the swarm's lossy communications environment, and
- identification of requirements for successful MLS utilization in these environments.

The remainder of this paper is organized into four sections. Section 2 provides a summary of related work on swarm system architectures and secure swarm communications. Section 3 describes the MLS implementation to include an overview of MLS, the Cisco C++ MLS application programming interface (API), and the existing ARSENL on-vehicle architecture into which MLS was incorporated. Section 4 discusses the results of testing the MLS implementation and analyzes its performance as a function of swarm size and key update rate. Finally, a summary of outcomes and implications of this work is provided in Section 5.

2. Related work

As robotic swarms have emerged as viable options for a variety of problems, increasing research has been conducted into the development of effective communication architectures.

In one effort, researchers from the Army Engineering University of the Chinese People's Liberation Army and Chengdu University [3] proposed supporting military swarm operations with a hybrid architecture combining a software-defined network (SDN) and Message Queue Telemetry Transport (MQTT), a publisher-subscriber middleware for devices characterized by computation- and communication-constrained networks, low processing, and low memory. At the time of publication, however, the researchers had not implemented the network structure in a real environment.

In 2015, Rosati et al. [4] proposed a flying ad-hoc network (FANET) in which the Optimized Link State Routing (OLSR) protocol was extended to use GPS data to predictively improve routing. Experiments with a system comprised of two fixed wing UAVs and a single ground station indicated that predictive OLSR responded more quickly to topology changes than than standard OLSR and that it did so without interruptions. The research did not specifically address communications security within the swarm.

Cellular network infrastructure has also been suggested as appropriate for swarm communications. Researchers from the University of North Dakota [5] proposed a UAV swarm architecture in which UAVs communicate among themselves over a cellular network without the need for a ground control station. This approach differs from a typical FANET in that routing is a function of the cellular infrastructure rather than the swarm system itself. It also potentially allows the UAVs to be dispersed across a larger area without sacrificing reliable data transfers. In addition, this approach allows some security concerns to be addressed at the infrastructure level, however it would be unlikely to fully satisfy the requirements of more sensitive information without additional application-layer features [6].

A two-phase communication protocol relying on both device-to-device communications and existing 4G and 5G cellular networks has been suggested by Han et al. [7]. In the first phase of their suggested protocol, ground control stations transmit a control message to all vehicles simultaneously over a cellular network. In the second phase, all vehicles that received the control message forward it to other vehicles in the swarm using device-to-device communications. The researchers concluded that this two-phase transmission protocol provided both high reliability and low latency for communications within the swarm. Again, communications security was not addressed.

All of these efforts focused on the capability of the communications architecture rather than its security. In effect, the suggested approaches implicitly assume that security requirements are handled by the infrastructure (layer 2 mostly–802.11, cellular, etc.) or that security features will be implemented on top of the communications architecture at the application layer.

3. MLS implementation on the ARSENL system

The Messaging Layer Security (MLS) cryptographic protocol was specifically designed with efficiency in mind and can potentially mitigate some of the issues identified in existing research. This section provides a brief overview of MLS and describes its integration into the ARSENL codebase using the open source C++ API developed by Cisco Systems.

3.1. MLS overview

The MLS standard is the product of an Internet Engineering Task Force (IETF) working group tasked with designing a protocol for secure group messaging [8]. MLS builds upon previous efforts in messaging security that have proven both effective and efficient [9, 10]. Like many of these approaches, MLS maintains group keys in a binary tree as depicted in Figure 1 where individual members are represented by leaf nodes. Each non-leaf node can contain a public key or may be blank (a byproduct of the update process). For any non-blank node, the associated private key is known to members occupying descendant leaf nodes [8]. Thus, the leaf node to which a member is assigned determines the subset of private keys that it knows.

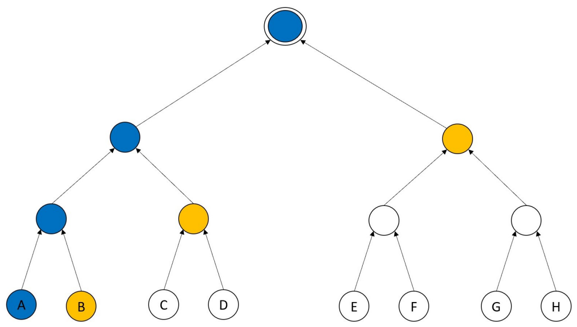


Figure 1. A MLS group tree example depicting member A's direct path (blue) and co-path (yellow) [11].

The binary tree allows for efficient updates. A member conducts an update by first generating a new public-private key pair for its leaf node and then iteratively generating key pairs for nodes in its direct path (depicted in blue in Figure 1). Private keys are shared with other group members using the public keys in the updating member's co-path nodes (depicted in yellow) [12]. As of Draft 7, MLS conducts these operations using the TreeKEM$_B$ protocol [8].

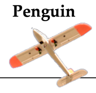
There are three major operations associated with MLS [8]. An `Update` operation is used by a member to change its own keys and those along its direct path. This can be seen as changing the blue nodes on Figure 1, such that the group key is updated. An `Add` is initiated by an existing group member to bring a new member into the group, supplying it with the current group state. Finally, a `Remove` is initiated to voluntarily or forcibly eject a member from the group, an action that is solidified by updating the group key while not providing the ejected member information on the update. All operations are executed through a two-step process in which the initiating member first sends a `Proposal` message indicating the operation type and follows up with a `Commit` message containing the updated key information, which cements the operation action. No changes are made to the tree until the `Commit` message has been processed.

There has been extensive research on MLS [12, 13, 14], and the MLS working group has endorsed API implementations available in C++, Rust, Go, and TypeScript [15]. The Cisco-developed C++ API [2] was used for the ARSENL swarm implementation because of the ease with which C++ can be incorporated into the existing on-vehicle software. The Cisco C++ API version that was utilized in this work implements MLS Draft 11. It is referred to as MLS++ for the remainder of this paper.

3.2. The ARSENL swarm system

The ARSENL multi-UAV system into which MLS was incorporated is composed of three UAV platform types: the Ritewing *Zephyr II* blended-wing UAV, the ARSENL-developed *Mosquito Hawk* quadcopter UAV, and the Finwing *Penguin* mid-wing pusher propeller UAV [16]. Each platform is equipped with a PixHawk-family autopilot and a Hardkernel Odroid-family companion computer. The basic specifications and operational parameters for the three platforms are shown in Table 1. Variant selections of these were used in simulation and physical testing, as is described later.

Table 1. ARSENL UAV Platform Configuration [16]

	Zephyr II	Mosquito Hawk	Penguin 
Dimensions	1.45m (wingspan)	0.29m (motor axis to motor axis)	1.73m (wingspan)
Maximum endurance	50 min	20 min	45 min
Cruise speed	18m/s	15m/s	12m/s
Weight	2.5kg	0.64kg	1.8kg
Autopilot computer	Pixhawk	PixRacer R15	PixRacerR15
Autopilot software	ArduPlane 3.8	ArduCopter 3.5	ArduPlane 3.8
Autonomy companion computer	Odroid U3	Odroid C0	Odroid C0

The ARSENL on-vehicle swarm functionality is implemented on the companion computer as a set of related Robot Operating System (ROS) [17] processes. ROS provides Python, C++, and Lisp APIs. The ARSENL implementation is written in Python; however, the C++ API [18] was used for this research to facilitate use of the MLS++ API.

Each ARSENL process serves a specific purpose in the architecture and runs as a ROS *node*. ARSENL nodes with which the MLS implementation interacts include the *network_bridge* node that manages current air-to-air and air-to-ground communications and the *task_scheduler* node that initiates preplanned actions as mission milestones are achieved.

Inter-node communication relies on a ROS publisher-subscriber model in which nodes publish messages to named topics to which other nodes can subscribe. ROS *messages* are typed data structures that can be used to encode atomic data elements (e.g., integers, floating point numbers, or strings) or composite data elements (i.e., structs) [17]. ROS utilizes the system's underlying network implementation to exchange messages between nodes as User Datagram Protocol (UDP) packets.

Inter-UAV communications take place over an 802.11n ad hoc network using an ARSENL-developed application-layer protocol. All messages are sent as UDP messages to the network broadcast address. With this architecture, all vehicles potentially receive every message, but receipt by any UAV is not guaranteed.

Over the course of a mission, ARSENL vehicles transition through a series of mission states [19]. In the *preflight* and *flight ready* states, the vehicle is prepared for launch. Once launched, the vehicle transitions to the *ingress* state in which it travels through a series of preplanned waypoints to the staging area. Upon arrival at the staging waypoint, the vehicle transitions to the *swarm ready* state from which swarm behaviors are executed. Finally, *landing* and *post flight* stages are used to recover the vehicle upon completion of swarm operations. State transitions are significant from the standpoint of this work in that transition to the *flight ready* state is the point at which the MLS group is initialized or joined, and the join process itself requires the identification of another UAV in the *flight ready*, *ingress*, or *swarm ready* state with which to initiate the join process. The join or initialize process is initiated automatically by the joining UAV's *task_scheduler* node.

3.3. Integration of MLS as a ROS node

The MLS++ API was installed alongside the ARSENL codebase within the ROS directory hierarchy, and the static libraries were automatically linked by the ROS *catkin_make* build tool at compilation. The API was used to develop a single **MLS** class, an instance of which is instantiated by a *mls* ROS node on each vehicle. The implementation is summarized here and described in detail in [11].

The ARSENL **MLS** class instance relies on three MLS++ classes: **Client**, **Session**, and **PendingJoin**. It uses a **Client** class instance to create or join the MLS group and to maintain key and ciphersuite information. The **PendingJoin** class instance is used to generate key packages and execute join-specific operations. After joining a group, an instance of the **Session** class manages the member's ongoing participation in the group.

3.3.1. Implementation overview Four ROS topics were added to the existing ARSENL runtime architecture as depicted in Figure 2 to facilitate interaction between the *mls* node and other ARSENL nodes. MLS-related information exchanged between nodes consists of unsigned integers (i.e., **std_msgs/UInt8**) and byte arrays containing serialized plaintext ARSENL messages (i.e., **arsenl_msgs/PackedNetworkMessage**). Publication or subscription to the ROS topic is indicated by the figure's arrows.

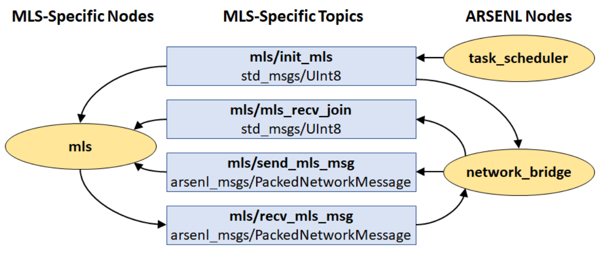


Figure 2. MLS ROS node integration into the ARSENL on-vehicle architecture [11].

The **MLS** class instance manages all MLS group operations. UAV-specific information such as UAV ID, network device, and network port is maintained as ROS parameters that are available to all nodes. The UAV ID is a unique integral value that serves as the vehicle's identification in the MLS group. The network device name and port identify the socket pair that is used for inter-UAV MLS communications.

UAVs initialize or join the MLS group upon reaching the *flight ready* state. The first UAV initializes the MLS group, and subsequent UAVs request to join the already established group. We pre-assume the trust structure for MLS identity keys, for simplicity, and leave the authentication of such keys (e.g. via certificates) as out-of-scope.

Both establish and join events are triggered by publication of a message to the *mls/init_mls* ROS topic by the *task_scheduler* node when the UAV transitions to the *flight ready* state. An integer 0 in the message's data field indicates that the UAV is to create a new MLS group. A non-zero value means that the UAV is to join an existing group and indicates the ID of the UAV that will execute the join operation on behalf of the new member.

The determination of whether to create a new group or join an existing group is based on the contents of unencrypted message traffic that is required for safety of flight and operator oversight. This traffic includes messages that enable the *task_scheduler* node on the joining UAV to identify an established UAV with which to initiate the join process or to determine that the group has not yet been established.

All established members of the group can encrypt messages to other group members and can decrypt received messages. The *network_bridge* node publishes ARSENL messages to be encrypted to the *mls/send_mls_msg* topic. The message-handling callback in the *mls* node encrypts the messages and broadcasts them over the MLS socket. Messages received from other UAVs are decrypted by the *mls* node and published to the ROS *mls/recv_mls_msg* topic for processing by the *network_bridge* node.

Updates and removals are initiated by the *mls* node as required. The initiating UAV sends the update or remove operation `Proposal` and `Commit` messages to all other UAVs over the MLS socket. Receiving UAVs process those messages to update their local MLS trees and ensure view consistency among the group.

These processes are described in more detail in the following sections.

3.3.2. MLS network communications A UDP broadcast socket is used in this implementation for all MLS network traffic, and all messages are sent to the broadcast address. Since much of the ARSENL code relies on UDP broadcast communications already, it made sense to continue with this pattern. This architecture does result in every message potentially being received by all UAVs. Therefore, messages intended for a specific recipient (e.g., as part of the join process) must be differentiated by their payload and processed accordingly upon receipt by the *mls* node. It is also important to note that UDP provides no guarantee of delivery, and neither the existing ARSENL codebase nor the *mls* node implementation checks to make sure that messages are received. The impact of this architecture on MLS performance is discussed in Section 4.

The asynchronous nature of MLS communications is accounted for by the implementation of a dedicated *mls* node thread for reading from the MLS socket. Writing to the socket occurs primarily within ROS topic callback functions. The callback for the *mls/send_mls_msg* topic, for instance, writes encrypted messages to the socket. Callback functions are invoked from within a specific ROS message-handling thread in response to messages received from subscribed topics [17].

MLS-specific data, such as encrypted messages, `KeyPackages`, `Proposal` and `Commit` messages, and `Welcome` messages are stored in an MLS++ **Bytes** object that encodes its contents as a vector of unsigned chars (i.e., bytes). Because data is written to and read from a socket as an array of unsigned chars, serialization of the **Bytes** object is required to send data and deserialization is required to convert received data back into a **Bytes** object.

Message serialization necessitated specific features to ensure that data was sent over the network in a manner that enabled it to be reassembled in a meaningful way. When a **Bytes** object is serialized, metadata associated with the object, including the type of message, is lost. In order for it to be deserialized successfully upon receipt, at least some of the associated metadata was included in the serialization.

The MLS++ API differentiates between message types through subtyping. That is, messages encoded as specific **Bytes** class subtypes can be handled correctly through polymorphism. The subtype is lost, however, when the object is converted to a char array. To account for this, our implementation adds a single element to the front the serialized array to serve as a packet type indicator that is checked during deserialization.

3.3.3. Proposal and Commit messages MLS operations are executed through a two-step process in which a `Proposal` message is used to specify the nature of the operation and a `Commit` message is used to finalize the operation and provide the information necessary to update the group members' local MLS trees. Group members can process `Proposal` messages in any order, but every UAV must process `Commit` messages in the same order to ensure consistent views among the group.

3.3.4. Group creation and member addition In the ARSENL MLS implementation, the first UAV to transition to the *flight ready* state, creates the group. As subsequent UAVs transition to the *flight ready* state, the *task_scheduler* node identifies another *flight ready*, *ingress*, or *swarm ready* UAV, and the *network_bridge* node sends an unencrypted ARSENL message to that UAV requesting to join the group.

The join process consists of a series of interactions between the joining and established UAVs as depicted in Figure 3. Upon receipt of the join request, the *network_bridge* node on the established UAV publishes a message containing the ID of the joining UAV to the *mls/mls_recv_join* ROS topic. The message-handling callback in the *mls* node initiates a handshake process with the joining UAV by sending a **Handshake** message. Both the join request and handshake are an addenda to the MLS protocol, which we incorporate to synchronize the join within the ARSENL framework to ensure that messages are processed in the correct order [11] (denoted by font difference). Once initiated, the join process is completely implemented in the joining and established UAV *mls* nodes (i.e., it does not involve any ARSENL nodes).

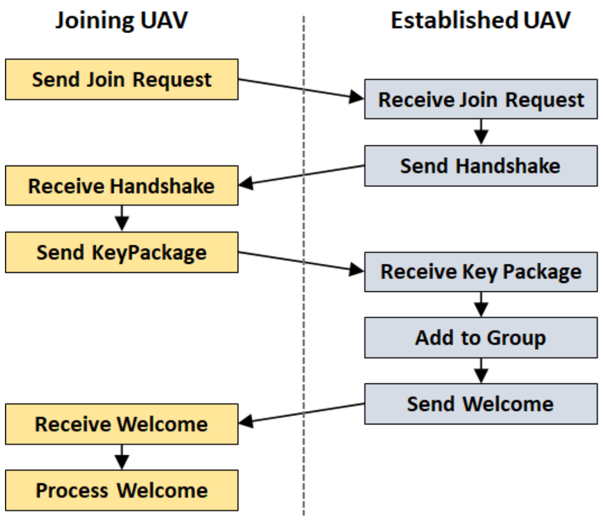


Figure 3. The ARSENL swarm MLS join process.

When the joining UAV receives the **Handshake** message, it prepares a `KeyPackage` containing its public key (among other information) and sends it to the established UAV [8]. Upon receipt of the `KeyPackage`, the established UAV prepares and broadcasts an add `Proposal` message and a `Commit` message to all other UAVs to add the new member. Group UAVs process these messages to update their local group tree views. Following addition of the new member to the existing group, the established UAV sends a `Welcome` message containing information regarding the current state of the group and any required public and private keys to the new member [8]. The joining UAV processes the `Welcome` message to finalize its addition to the group.

3.3.5. Sending and receiving encrypted messages

As indicated in Figure 2, the *mls* node subscribes to the *mls/send_mls_msg* ROS topic. When an ARSENL message is to be encrypted and sent to one or more group members, the *network_bridge* node serializes the plaintext message and publishes it to this topic. The *mls* node callback function encrypts the message and broadcasts the encrypted data over the MLS socket. When a UAV receives an encrypted message, it is decrypted and published to the *mls/recv_mls_msg* ROS topic by the *mls* node's socket-read thread. From there, the *network_bridge* node callback function processes (i.e., deserializes) the message and forwards the plaintext contents for use by other ARSENL nodes.

As implemented, the *mls* node only encrypts some ARSENL network traffic, namely the intra-swarm traffic. Information exchanged between UAVs and the ground station that would normally be required to ensure safety of flight and maintain operator oversight is not encrypted as swarm traffic. As noted earlier, this unencrypted traffic includes the ID and swarm state information that is used by the *task_scheduler* node to identify a UAV to perform the join operation. In our tests, this is left unencrypted as an out-of-scope communication link, but in a normal operating scenario the channels from devices to the operator might be encrypted under a different protocol. It is the information that is transmitted between UAVs that is encrypted using MLS. In particular, state/telemetry messages transmitted by each UAV at a rate of eight hertz and autopilot status messages transmitted at one hertz are encrypted. Since these messages comprise the bulk of the ARSENL network traffic [19], performance observations are considered representative of the overall system.

3.3.6. Update

The update operation is conducted periodically to ensure that security of the group is maintained. In particular, individual and group keys are updated according to a key schedule in order to provide forward secrecy and post-compromise security [8]. When a UAV wants to initiate an update, it sends an update `Proposal` message to the group and follows that with a `Commit` message with the specific update information. Each UAV that receives the messages processes them to update the keys in their local view of the tree.

3.3.7. Member removal

The last MLS operation tested was removal of a member. This operation is required when a member voluntarily leaves the group or when other group members determine that the member has failed, has been compromised, or is malfunctioning. The removal process is similar to the update operation in that it is effected by `Proposal` and `Commit` messages from one member to the rest of the group. In addition to deleting the removed member's node from the MLS tree, all nodes in the removed member's direct path must be blanked or updated. Once this occurs, the removed member will no longer be able to decrypt messages exchanged between the remaining group members.

4. Results

Testing was conducted in two stages. The first tests were conducted in the ARSENL software-in-the-loop (SITL) simulation environment [20] to provide baseline metrics and assurances of correct functionality. The second phase consisted of ground tests with various numbers of ARSENL UAVs and was used to assess performance in the actual communications environment. Live vehicle testing was also used to evaluate various update intervals in the lossy communication environment in the absence of delivery service implementation. Testing results are summarized here. Exhaustive test results and further analysis is available in [11].

4.1. Simulation system testing

All three of the MLS operations described in Section 3.3 were tested in the SITL environment and were found to be correctly implemented. SITL environment testing included both fixed-wing Zephyr II and quadrotor Mosquito Hawk UAVs and included transition through all of the swarm states and execution of swarm behaviors. Five variables of interest were specifically assessed during SITL environment testing to evaluate MLS performance generally: 1) update frequency, 2) bytes of plaintext encrypted and sent, 3) bytes of ciphertext received and decrypted, 4) number of messages encrypted and sent, and 5) number of messages received and decrypted. Statistics were captured and logged every three seconds.

Testing was conducted with various update intervals associated with the number of messages sent by each UAV, meaning an update is initiated by the UAV for every x messages sent. Testing was conducted over time periods ranging from 30 seconds to 10 minutes, and data for each experiment was aggregated into a single per-second rate. Data was collected for update intervals of $x = 50$ messages, 150 messages, and 250 messages per device. For these intervals, each participant initiated an update approximately every five seconds, 15 seconds, and 25 seconds respectively. Thus, the group state (e.g., shared group encryption key) was updated more frequently as the number of UAVs in the swarm increased. Data was also collected with no updates being initiated by the participating UAVs.

Every update interval test experienced at least one UAV that failed to join or dropped out of group communications. Empirical evidence (i.e., log entries and ROS messages to the terminal) suggests that these failures were caused by unprocessed updates and other errors associated with missed `Commit` messages. Update frequency testing was not exhaustive with respect to an optimum update interval and should be more thoroughly investigated in future research. The MLS specification requires a distribution service that guarantees reliable transport; since one was not implemented, such a service for MLS maintenance messages will likely be required before sufficient testing can be conducted to fully assess update interval options.

Decrypted messages per second results from the SITL environment update interval tests are depicted in Figure 4. Results for most intervals for a five-UAV swarm are close to the expected value of 45 decrypted messages per second per UAV. Missed updates were evidently not an issue for small swarms. For the 10-UAV swarm, however, the update interval significantly impacted the average number of messages decrypted per second. Not surprisingly, the swarms that had no updates and the swarms with the 250-message update interval outperformed swarms for the other intervals tested. With an update interval of 50 messages, the 10 UAVs received and decrypted fewer than half of the messages that were sent. The average number of messages decrypted per second got closer to the expected value as the update interval increased, indicating that the swarm performs better with a longer period between updates. This result is not surprising since update operations impose both networking and processing overhead, and increasing the frequency of updates increases the number of opportunities for UAVs to miss `Commit` messages in absence of a reliable distribution service.

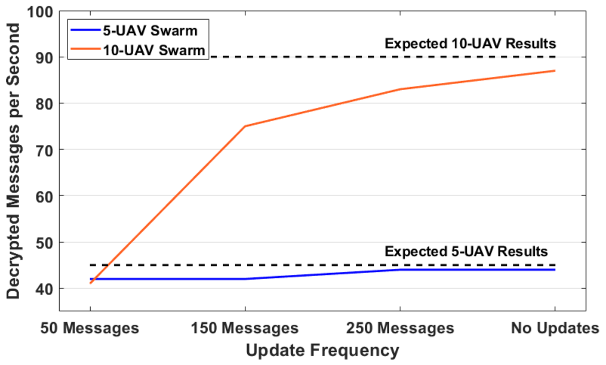


Figure 4. Observed and expected decrypted messages per second for groups of five and 10 UAVs for software-in-the-loop simulation tests [11].

The improvement as the interval increased from 50 to 150 messages and from 150 messages to 250 messages implies that even longer intervals might yield results equivalent to those of the no update test. Even occasional unprocessed updates would be problematic in an operational system, however, since affected UAVs would not be able to communicate with the group beyond the point of the missed update. Over time, the number of UAVs that successfully process all updates will decrease even for long update intervals. However, when update intervals are aligned session length in, e.g. 802.11, this appears less problematic, and the relative security gains for update intervals shorter than normal sessions is notable.

In addition, it is reasonable to infer from the decrease in performance between the five-UAV and 10-UAV tests that an interval sufficient for a 10-UAV system is unlikely to perform well as size increases further. Security concerns associated with longer update intervals will also need to be addressed.

Since the 250-message-per-device interval worked reasonably well, only this interval and no update processing were tested on physical aircraft in ground tests.

4.2. On-vehicle testing

Following testing in the SITL environment, ground tests were conducted with physical UAVs to test the functionality of the *mls* node in the actual ARSENL vehicles (flight testing is anticipated once satisfactory results are observed in ground tests). All testing was conducted on the ARSENL-developed Mosquito Hawk quadcoptor [21]. This platform was chosen because it has the least powerful payload computer of all of the ARSENL UAVs. Potential issues, therefore, are more likely to manifest on the Mosquito Hawk (or the identically configured Penguin) than on the more powerful Zephyr II payload computer.

Mosquito Hawk autonomy and communications functionality is implemented on a HardKernel Odroid C0 companion computer, a single-board computer with an Advanced Reduced Instruction Set Computer Machine (ARM) version 7 quadcore chipset that operates at 1.5 gigahertz. The C0 has one gigabyte of random access memory and a 32 gigabyte embedded MultiMediaCard (eMMC). Inter-UAV communications take place over an 802.11n network in the 2.4 gigahertz band as described in Section 3.2 [19].

Update issues identified in the simulation environment were taken into account when testing with the physical UAVs. A main focus with the ground testing was to determine if the *mls* node worked properly on physical aircraft (i.e., to confirm that UAVs could join a group and encrypt and decrypt messages to and from other members). Data was collected from on-vehicle log files to which summary data was written every three seconds. Collected data includes bytes of data encrypted and decrypted, number of messages encrypted and decrypted, and the time in milliseconds required to join the swarm.

Two types of MLS-related errors were encountered during the ground tests:

1. failure to join the MLS group and
2. failure to process join or update operation `Commit` messages.

These failures were all related in some way to missed MLS messages (possibly resulting from UDP unreliability), which in turn affected how MLS functioned within the swarm. Reliable receipt of MLS maintenance messages is an important requirement that future work will need to address.

4.2.1. No MLS update performance Ground tests were first conducted with UAV swarms that were not issuing MLS updates. Four different swarm sizes were tested: three members, five members, seven members, and 10 members. Both failure types described above were observed during the ground tests with no updates. This is not surprising since UAVs still need to process `Commit` messages associated with joins even when no update operations are being processed. If the lossy communications environment results in a UAV missing one of these `Commit` messages, its local view of the group will lose synchronization with the rest of the group. As a result the affected UAV will no longer be able to communicate with the rest of the group using the updated group keys.

Figure 5 shows the results of the ground tests conducted with no updates. The blue line shows that as the swarm size grows, the average number of messages decrypted per second per UAV grows approximately linearly. This is expected behavior from the swarm and supports a promising outlook for the future use of MLS for secure swarm communications.

4.2.2. 250 message MLS update interval After testing swarms with no updates, a 250-message-per-device interval between updates was introduced. Based on the typical message throughput (i.e., nine ARSENL messages per second), this interval equates each vehicle initiating an update approximately every 28 seconds. This interval performed the best in the SITL environment tests, so it was assumed to be the most likely interval to perform well on the physical UAVs. Results indicate that the much more demanding

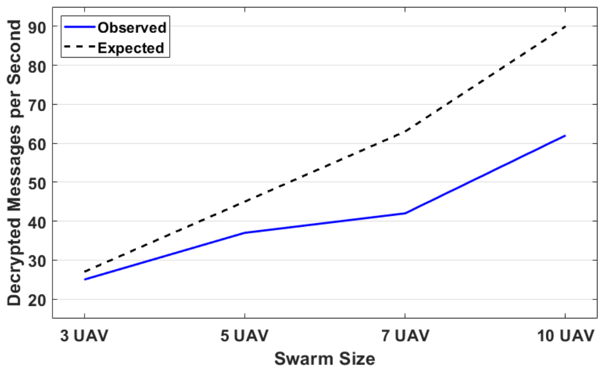


Figure 5. Observed and expected decrypted messages per second for ground tests with no MLS updates [11].

real-world communications environment significantly impacted the ability to conduct scheduled MLS updates even at this interval, however.

Swarm sizes of three, four, five, seven, and 12 were tested for this interval, but unreliable MLS message receipt made testing with updates difficult. Communications issues among the UAVs led to multiple ground tests that yielded no data for any UAVs in the group. For the tests of swarm size four and seven, for instance, not a single UAV produced a log file, indicating a failure to join the group. Failure to join the group can result from one of two failures. First, a UAV can get stuck during the handshake if a packet to or from the adding UAV is not delivered. It is also possible for a UAV to send the join request to a UAV that is not an active group participant (i.e., either its own join was unsuccessful or it missed an update). This second failure is partially a result of the ARSENL implementation in which the *task_scheduler* node assumes that any UAV that has reached the *flight ready* state is a member of the group. This cascading effect of UAVs failing to join the group was not accounted for in this implementation but should be addressed in follow-on work.

Figure 6 summarizes the results of the 250-message-per-device update interval tests. Even among UAVs that successfully joined the group, the results indicate that as swarm sizes increased and more update operations were processed, UAVs were more likely to miss processing a `Commit` message and lose communication with the rest of the group. This graph shows that as the swarm size increased, fewer messages were decrypted on average, indicating that few UAVs maintained synchronization with the common group keys for the duration of the tests. As with the SITL environment tests, this performance can be expected to degrade even further over time as more UAVs fail to process updates. UAV-specific logs in which some vehicles processed more encrypted messages than others or only decrypted messages for a portion of the experiment reinforce this observation.

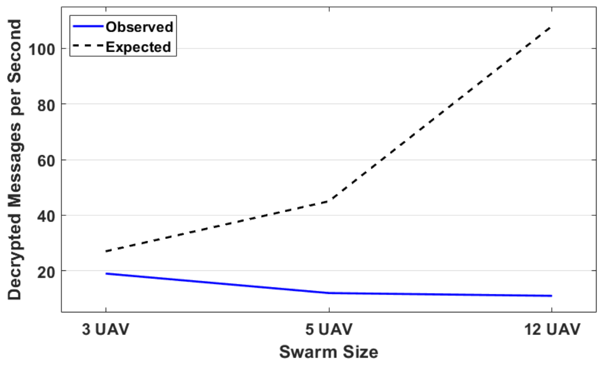


Figure 6. Observed and expected average decrypted messages per second for ground testing with a 250-message MLS update interval [11].

In-vehicle results clearly indicate that a reliable delivery service is required for MLS to work with a lossy-communications swarm system such as ARSENL's. Both the join operation and the update operation are heavily dependent on receipt of all MLS maintenance messages.

Overall, the SITL environment and real-world test results are encouraging in that they demonstrate the ability of the swarm to successfully employ MLS for communications security. On the other hand, the difficulty when updates were included or swarm size increased highlight the susceptibility of MLS to situations in which reliable delivery of MLS messages cannot be assured.

5. Conclusion

This work provides a proof of concept for using MLS to provide secure communications within a swarm of small UAVs. Results provide evidence that MLS can perform well in a swarm and that swarm platforms are capable of satisfying MLS computational requirements without impacting swarm performance. However, the results also highlight the need for future work on mitigations for the unreliable transport channels upon which these systems often rely.

Most shortcomings identified with the ARSENL MLS implementation are associated with join and update operations, which coincide with key rotation. Given the lossiness and frequent segmentation that often characterize swarm communications architectures, UAVs can easily get out of synchronization with the rest of the swarm. Once this occurs, there are few good options available to repair the group. This loss of key synchronization contributed to some limitations in the testing and verification of the *mls* node in the ARSENL swarm.

The testing conducted here did indicate that a longer period between updates works better with the swarm architecture because of reduced overhead and vulnerability to network limitations. Longer intervals between updates do leave the swarm open to some period of vulnerability if a UAV is compromised; however, such periods may still be several magnitudes of measure shorter than update intervals of an 802.11 session for example. Inclusion of a distribution service that can ensure reliable delivery of MLS maintenance messages may make it possible to conduct updates more frequently. The MLS protocol does call for a distribution service when used with unreliable networks, however a swarm-suitable system was not available for testing. Future research will need to include the development of a such a distribution service if MLS is to be successfully utilized with these systems. Regardless, update interval decisions will often come down to a trade-off between what the communications bandwidth will allow (particularly as swarm size increases) and what security the mission requires.

References

[1] R. Barnes, B. Beurdouche, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert, "The Messaging Layer Security (MLS) Protocol: draft-ietf-mls-protocol-16." `https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-16`, July 2022. Internet Engineering Task Force (IETF). Draft 16.

[2] Cisco Systems, Inc., "MLS++ Github repository." `https://github.com/mlswg/mls-protocol` [Accessed June 2022], 2022.

[3] F. Xiong, A. Li, H. Wang, and L. Tang, "An sdn-mqtt based communication system for battlefield uav swarms," *IEEE Communications Magazine*, vol. 57, no. 8, pp. 41–47, 2019.

[4] S. Rosati, K. Krużelecki, G. Heitz, D. Floreano, and B. Rimoldi, "Dynamic routing for flying ad hoc networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 3, pp. 1690–1700, 2015.

[5] M. Campion, P. Ranganathan, and S. Faruque, "A review and future directions of uav swarm communication architectures.," *EIT*, pp. 903–908, 2018.

[6] R. B. Thompson and P. Thulasiraman, "Confidential and authenticated communications in a large fixed-wing UAV swarm," in *IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pp. 375–382, October 2016.

[7] Y. Han, L. Liu, L. Duan, and R. Zhang, "Towards reliable uav swarm communication in d2d-enhanced cellular networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1567–1581, 2020.

[8] R. Barnes, B. Beurdouche, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert, "The messaging layer security (mls) protocol," *Internet Engineering Task Force, Internet-Draft draft-ietfmls-protocol-11*, 2020.

[9] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, "On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1802–1819, 2018.

[10] K. Bhargavan, R. Barnes, and E. Rescorla, *TreeKEM: asynchronous decentralized key management for large dynamic groups a protocol proposal for messaging layer security (MLS).* PhD thesis, Inria Paris, 2018.

[11] E. Dietz, "Utilizing the messaging layer security protocol in a lossy communications aerial swarm," MS thesis, Naval Postgraduate School, Monterey, CA, April 2022.

[12] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis, "Security analysis and improvements for the IETF MLS standard for group messaging," in *Annual International Cryptology Conference*, pp. 248–277, Springer, 2020.

[13] K. Klein, G. Pascual-Perez, M. Walter, C. Kamath, M. Capretto, M. Cueto, I. Markov, M. Yeo, J. Alwen, and K. Pietrzak, "Keep the dirt: Tainted treekem, adaptively and actively secure continuous group key agreement," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 268–284, IEEE, 2021.

[14] C. J. F. Cremers, B. Hale, and K. Kohbrok, "The complexities of healing in secure group messaging: Why cross-group effects matter," in *USENIX Security Symposium*, 2021.

[15] MLS Working Group, "MLS working group–mls implementations Github repository." `https://github.com/mlswg/mls-implementations` [Accessed June 2022], 2021.

[16] M. S. Hopchak, D. T. Davis, K. B. Giles, K. D. Jones, and M. J. Jones, "Autonomous area search using market-based assignment in multi-vehicle unmanned systems," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, June 2022.

[17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, 2009.

[18] Open Robotics, "ROS wiki–roscpp." `http://wiki.ros.org/roscpp` Accessed June 2022 [Online].

[19] T. H. Chung, M. R. Clement, M. A. Day, K. D. Jones, D. Davis, and M. Jones, "Live-fly, large-scale field experimentation for large numbers of fixed-wing uavs," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1255–1262, IEEE, 2016.

[20] M. A. Day, M. R. Clement, J. D. Russo, D. Davis, and T. H. Chung, "Multi-UAV software systems and simulation architecture," in *2015 International Conference on Unmanned Aerial Systems*, pp. 426–435, IEEE, 2015.

[21] K. B. Giles, D. T. Davis, K. D. Jones, and M. J. Jones, "Expanding domains for multi-vehicle unmanned systems," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1400–1409, IEEE, 2021.