cryptography

MDPI

Article

Securely Scaling Autonomy: The Role of Cryptography in Future Unmanned Aircraft Systems (UASs)

Paul Rochford, William J. Buchanan *, Rich Macfarlane and Madjid Tehrani

Blockpass ID Lab, Edinburgh Napier University, Edinburgh EH10 5DT, UK
* Correspondence: b.buchanan@napier.ac.uk

Abstract

The decentralisation of autonomous Unmanned Aircraft Systems (UASs) introduces significant challenges in terms of establishing secure communication and consensus in contested, resource-constrained environments. This research addresses these challenges by conducting a comprehensive performance evaluation of two cryptographic technologies: Messaging Layer Security (MLS) for group key exchange, and threshold signatures (FROST and BLS) for decentralised consensus. Seven leading open-source libraries were methodically assessed through a series of static, network-simulated, and novel bulk-signing benchmarks to measure their computational efficiency and practical resilience. This paper confirms that MLS is a viable solution, capable of supporting the group sizes and throughput requirements of a UAS swarm. It corroborates prior work by identifying the Cisco MLSpp library as unsuitable for dynamic environments due to poorly scaling group management functions, while demonstrating that OpenMLS is a highly performant and scalable alternative. Furthermore, the findings show that operating MLS in a 'key management' mode offers a dramatic increase in performance and resilience, a critical trade-off for UAS operations. For consensus, the benchmarks reveal a range of compromises for developers to consider, while identifying the Zcash FROST implementation as the most effective all-around performer for sustained, high-volume use cases due to its balance of security features and efficient verification.

Keywords: unmanned aircraft systems; MLS framework; distributed key generation

1. Introduction

Modern distributed systems, for example, the Internet of Things and autonomous vehicle networks, have requirements for decentralisation, security, trust and consensus that often conflict with each other. Traditional approaches to security often rely on centralised authorities for key management and identity verification, which are ill-suited to environments consisting of power-limited and intermittently connected devices. The use of centralised authorities introduces single points of failure, bottlenecks and high-value targets for attack—undermining the resilience that distributed systems can offer [1–3].

Unmanned Aircraft Systems (UASs) are aircraft that operate without a pilot or crew on board. They are frequently referred to as Remotely Piloted Aircraft Systems (RPASs), Unmanned Aircraft Vehicles (UAVs) or drones, with the preferred nomenclature being dependent on the organisation operating or developing the system. These devices were initially piloted aircraft retrofitted for remote control and used for high-risk or one-way operations [4]. As technology improved, their use for reconnaissance and electronic warfare became more widespread, with armed platforms becoming prevalent during the Global

Academic Editors: Josef Pieprzyk and Aleksandr Ometov

Received: 16 November 2025
Revised: 11 March 2026
Accepted: 13 March 2026
Published: 20 March 2026

War on Terror and the invasions of Iraq and Afghanistan. These operations highlighted the security challenges involved in deploying relatively simple systems to an operational environment [5].

With improvements in battery, electric motor and radio frequency (RF) connectivity technologies, UASs have become accessible to consumers and non-military users. UASs now range from small devices that can be held in the palm of one's hand, with a range measured in metres, to traditional aircraft-sized platforms with a range measured in 1000s of km. Regardless of the scale, the key components of a UAS are an aerial vehicle and a ground control station. These are linked with an RF connection that provides control and data link capabilities [6]. Extensive research shows the vulnerability of these connections either to direct attack or as an attack vector to access the aircraft vehicle or control station [6,7]. The use of fibre-optic data links to remove this attack vector is a recent and well-documented operational example from the Russia–Ukraine war, which further highlights the importance of cybersecurity to UAS operations and the vulnerability created through the use of COTS products in a military capacity [8].

1.1. Motivation and Regulatory Context

The deployment of autonomous UASs within the UK is governed by a strict regulatory framework managed by the Civil Aviation Authority (CAA) and the Military Aviation Authority (MAA) [9,10]. Central to these regulations is a Secure by Design methodology, which mandates that cybersecurity and deterministic behaviour be integrated from the outset to prevent the manipulation of autonomous systems [11].

Furthermore, while modern autonomous technologies facilitate increased combat mass and operational reach, they must remain compliant with international laws regarding the responsible use of lethal weapons, often necessitating a "human in the loop" or verifiable consensus for mission-critical actions [12]. These requirements drive the technical need for the cryptographic architectures evaluated in this study: Messaging Layer Security (MLS) to provide resilient, authenticated group identities [13], and threshold signatures to enable verifiable, decentralised decision-making [14]. By moving away from centralised trust authorities, UAS swarms can better adhere to these regulatory mandates for resilience and safety in contested environments.

1.2. Contributions and Paper Structure

This paper investigates the role of applied cryptography in enabling secure and scalable autonomy for decentralised Unmanned Aircraft Systems (UASs). The contributions of this work are threefold. First, it provides a systematic evaluation of cryptographic protocols, specifically Messaging Layer Security (MLS) and threshold signature schemes, as practical enablers of secure group communication and decentralised consensus in dynamic and resource-constrained UAS networks. Second, it presents an empirical performance analysis of representative open-source implementations, benchmarking their computational, communication, and scaling characteristics under operationally relevant conditions. Third, the paper derives design-level insights for UAS architects by demonstrating how cryptographic configuration choices directly affect system resilience, scalability, and operational autonomy. Showing that certain MLS configurations offer substantial performance advantages, with important implications for UAS operating in contested electromagnetic environments.

The remainder of this paper is structured as follows: Section 2 provides the background and related work; Section 3 details the operational threat model and benchmarking methodology; Section 4 presents the experimental results; and Section 5 discusses the implications for autonomous UAS design.

2. Background and Related Work

The development of cryptographic protocols for autonomous UASs involves two primary functional challenges: secure, scalable group communication and decentralised consensus. This section explores the evolution of these primitives from foundational theory to modern implementations, with a focus on their application within distributed UAS networks.

2.1. Swarm Identity and Key Distribution

Establishing a secure communication "pipe" that can survive the loss or capture of individual nodes is fundamental to swarm resilience. Research into UAS secure communications has traditionally followed two distinct paths: hardware-dependent emerging technologies and software-based protocols.

Recent studies have explored hardware-based solutions, notably Physically Unclonable Functions (PUFs), to provide tamper-resistant authentication without the need for traditional certificate management [1]. While PUFs offer a lightweight "hardware fingerprint" resilient to physical key extraction, they typically operate at the device-to-device level and may face scalability challenges in dynamic swarms with high membership churn. Other approaches have utilised the Chinese Remainder Theorem (CRT) or Shamir Secret Sharing through centralised "trust authorities" or tethered UAVs [15,16]. While effective for static groups, these models introduce single points of failure or scale poorly as swarm size increases.

To address these limitations, an alternative approach is the use of Group-Authenticated Key Exchange (GAKE) mechanisms that provide logarithmic scaling ($O(\log n)$). The IETF-ratified Messaging Layer Security (MLS) protocol [13], built upon the Tree-based Key Encapsulation Method (TreeKEM) [17], is uniquely suited to this requirement. MLS provides robust group authentication and Post-Compromise Security (PCS), allowing for a swarm to cryptographically "heal" and exclude a compromised node from future mission epochs [18]. At least one study has evaluated whether the Rust-based OpenMLS can overcome the performance bottlenecks identified in earlier military UAS-specific implementations of MLS [19,20].

2.2. Decentralised Consensus and Threshold Cryptography

Beyond secure communication, swarms must make autonomous decisions without a persistent ground control station. This introduces the risk of Byzantine faults, where compromised or malfunctioning nodes transmit malicious commands to undermine the mission [21]. To achieve resilience, the system must ensure that no single node can authorise critical actions.

NASA's SAFE50 Autonomy Reference Architecture defines the extensive range of environmental hazards and failures that necessitate a consensus-based approach to autonomous UAS safety [22]. While policy frameworks highlight threshold signatures as a solution, there is a lack of empirical performance data for these protocols on embedded hardware. We bridge this gap by benchmarking two leading constructions:

1. BLS Signatures: A pairing-based protocol offering a single-round signing phase, minimising interactive latency at the cost of computationally intensive verification [23,24].
2. FROST: Flexible Round-Optimised Schnorr Threshold signatures maintain security with a reduced computational load compared to BLS, but require an additional communication round [14].

2.3. The Resource–Security Trade-Off

A critical theme in the recent UAS security literature is the impact of cryptographic overhead on system stability. In contested environments, CPU cycles are a finite survival resource, heavily burdened by electronic warfare (EW) countermeasures and flight-control tasks. Building on the research of Leon and Britt [19], this study evaluates the integration of MLS and threshold signatures as a cohesive stack. We hypothesise that the 20× performance gain provided by the MLS "key management mode" is not merely an optimisation but a mandatory requirement for maintaining the computational "headroom" necessary for operational resilience.

3. Methodology

This section applies concepts from the related works to the performance benchmarking of MLS and threshold signature libraries. These protocols may address two challenges relating to security within autonomous UAS—encryption key generation/distribution and consensus-based decision making. The benchmarking aims to identify suitable libraries for use in a related project and evaluate their performance under conditions relevant to a UAS network. A high-level outline is shown in Figure 1.

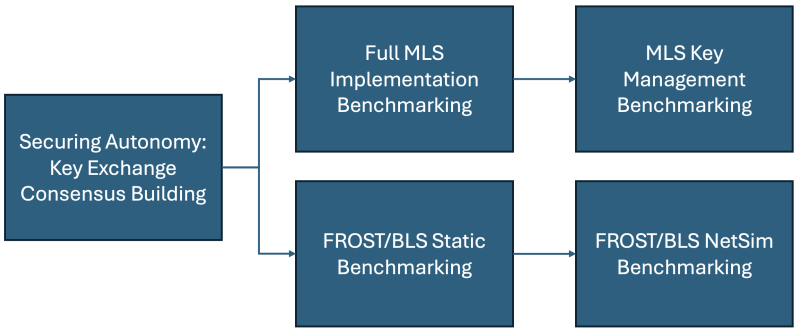

Figure 1. Research Scope.

This paper will focus on the computational efficiency of the libraries under test, the impact of network performance on their operation and the optimum way to utilise them. It will use a range of benchmarks that sequentially run through the operations of a protocol, allowing for their performance to be accurately measured. As the communication pattern for FROST and BLS varies, a dual approach of static and network-simulated benchmarks is used. In this way, the "computation time" and "communication time" can be separated and analysed [25–27].

3.1. UAS Concept of Employment

The wider project is developing a UAS cloud architecture that will create a meshed network with utility across a range of applications. The intention is for swarms ranging in size from 5–100 to be able to operate with significant autonomy, either through choice or as a result of lost C2 links. It is anticipated that individual UAVs will join or leave the swarm during a mission if a particular capability is required, and the network will be able to adapt gracefully to these changes.

3.2. Design Requirements

Although the project considers two different technologies/protocols, there are common design principles for both:

1. Decentralisation is a priority, but an initial centralised configuration phase is acceptable.
2. A majority of nodes will be available during the configuration phase.

3. Security must be maintained if a UAV is permanently lost.
4. A UAV can be temporarily out of communication and seamlessly rejoin the network.
5. The UAS network must function without a connection to the C2 or configuration node during normal operation.

3.3. Test Environment

The specification of the hardware solution for the wider project was not available prior to testing. Therefore, all the benchmark tests were run on a virtualised Linux system—Windows Subsystem for Linux (WSL) with a 12th Gen Intel(R) Core(TM) i5-12450H processor. The instance was allocated 12 logical processors (6 cores with 2 threads per core) and 7.6Gi of RAM. This configuration provided ample computational resources and memory for the test suites and ensuring the performance results were a valid measure of the libraries' efficiency.

3.4. Operational Threat Model

While the design requirements for this system prioritise decentralisation and graceful adaptation to membership changes, these requirements are fundamentally driven by the need for resilience against an active adversary. To provide a rigorous context for the performance benchmarks, we define the following threat environment:

1. Resilience Against Node Capture. In a contested environment, it is assumed that an adversary may physically capture a node and gain access to long-lived key material. This necessitates Messaging Layer Security (MLS) to provide Post-Compromise Security (PCS). By rotating group keys, the swarm can "heal" itself, ensuring stolen keys cannot be used to eavesdrop on future communications.
2. Resilience Against Byzantine Behaviour. Autonomous operation without a C2 link introduces the risk of "Byzantine" faults, where compromised nodes send malicious commands. This justifies the use of threshold signatures (FROST/BLS), ensuring no single node can authorize critical changes. A verifiable quorum (t-of-n) must agree, neutralising the impact of a "traitor" node.
3. Resilience Against Resource Exhaustion. In "worst-case" scenarios involving heavy signal jamming, CPU cycles become a finite survival resource. We evaluate the MLS key management mode as a defence against resource exhaustion; by reclaiming the 20× processing overhead of "Full Mode," we preserve the computational "headroom" required to survive external threats.

Operational Assumptions:

1. **Network Conditions:** We assume a degraded RF environment, modelled in our `NetSim` benchmarks with high packet loss and variable latency.
2. **Pre-Mission Trust:** Nodes are provisioned with initial root-of-trust credentials during a secure configuration phase.

3.5. MLS Benchmark Design

The requirement was for a protocol that could support group key distribution over 100 + clients in order to enable secure broadcast messaging required for threshold signature schemes (and other applications requiring encrypted messaging between nodes). The selected protocol needed to be well-proven from a security perspective, future-proof, and be able to support the addition and subtraction of clients during a mission. It should be capable of running with as much decentralisation as possible. The IETF MLS protocol has been shown to meet all these requirements.

3.5.1. High-Level Operation

Users of MLS are referred to as clients and are organised into groups and epochs. A group consists of two or more clients and the epoch defines the current chronological state of the group. The clients within a group agree on a common secret epoch key, using a tree-based sharing structure (Tree-based Key Encapsulation Method).

TreeKEM shares the epoch key, which is then used by each client to individually derive a 'secret tree'. This structure allows a client to locally derive a unique, symmetric key for each message they send, using a Key Derivation Function (KDF) with their branch of the secret tree and a message sequence number. Message encryption can either be done using the MLS framing layer or by exporting a key for use within another application. Messages sent through the framing layer are protected with a per-message encryption key and are also signed by the sender, assuring the confidentiality, integrity and authenticity of the message. Use of the key export function loses the additional assurance of per-message signatures, but allows applications that MLS does not support, such as streaming Full Motion Video (FMV), to still utilise the group AKE functionality.

TreeKEM solves the scalability problem faced by the trusted dealer model. It allows clients to hold keys for the nodes on its path to the root. When an update occurs, only a portion of the tree needs to be re-encrypted and re-distributed, not the entire key. This reduces the complexity of operations from linear to logarithmic with group size, making it far more efficient for large groups than a trusted dealer. The ability of an MLS library to realise these benefits and the comparative performance of different libraries is the focus of this research.

3.5.2. MLS Architecture

A simple MLS architecture is shown in Figure 2. The protocol relies on the availability of an Authentication Service (AS) and a Delivery Service (DS). The AS allows a client to authenticate credentials presented by another group member, or a proposed group member and the AS is assumed to be a trusted entity. The MLS RFC supports the use of PKC as an acceptable means of implementing a AS, and this approach is compatible with our pre-mission configuration phase concept.

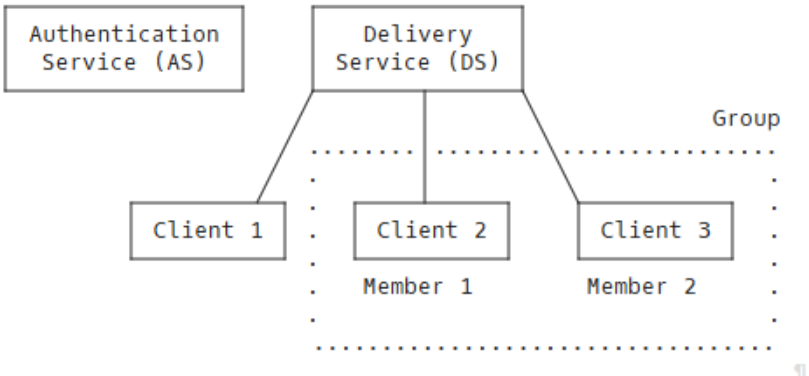

Figure 2. Example MLS Architecture [13].

The DS facilitates group state management by enforcing message ordering for membership changes initiated by the clients. It is also responsible for the scheduling of key updates (epoch changes) and for routing messages amongst the group for messages sent within the MLS framing layer. Within the MLS protocol message, ordering is critical for success, and the DS is the key component to enable that. When employed in a mobile messaging app, this would be a fixed server. This centralised approach could be replicated within our UAS architecture, with replication to backup DS providing resilience to a primary DS

becoming unavailable. However, the RFC also supports the concept of a decentralised peer-to-peer DS, and previous work by Marstrander [20] has shown this to be feasible. His implementation is publicly available as a proof of concept and would provide a starting point to develop a DS for a specific architecture and hardware configuration.

3.5.3. MLS Operational Drivers

The Messaging Layer Security (MLS) protocol, defined in IETF RFC 9420 [13], utilizes a complex arrangement of binary tree structures to provide scalable group security. To isolate the performance characteristics of the libraries under test, we categorized the eight primary MLS operations into three functional groups based on their computational and architectural impact (see Table 1).

Table 1. Drivers and Performance Implications.

Category	Operations	Technical Driver
Membership State	Add, Remove, Update	TreeKEM path updates ($O(\log n)$ scalability)
State Synchronization	Process Update	Decryption and local state reconciliation
Data Path (Full)	Protect, Unprotect	Per-message digital signing and framing overhead
Data Path (Optimized)	Key Export	Local KDF derivation via symmetric encryption

Membership State Transitions

This group—comprising *Add*, *Update*, and *Remove Member*—measures the efficiency of the Tree-based Key Encapsulation Method (TreeKEM) state machine. These operations are computationally intensive as they require modifying the binary tree structure and re-encrypting path secrets to the root to maintain forward secrecy and Post-Compromise Security (PCS) [13,17]. In a UAS swarm context, these represent the "identity churn" cost associated with ad hoc expansion or the revocation of captured nodes.

State Synchronisation

The *Process Update* operation measures the recipient-side cost of a group state change. Unlike the initiator operations, this tests the library's ability to efficiently decrypt incoming Commit messages and reconcile the local copy of the group tree [18]. This metric is vital for evaluating the "collateral" processing load placed on the entire swarm whenever a single node initiates a re-keying event.

Data Path and Throughput Modes

These operations—*Protect Message*, *Unprotect Message*, and *Key Export*—distinguish between the two primary modes of operation evaluated in this study. *Protect* and *Unprotect* measure the overhead of the standard MLS framing layer, which applies per-message digital signatures for attribution. Conversely, *Key Export* utilizes a local Key Derivation Function (KDF) to provide symmetric keys for external application layers [13] This grouping allows for a direct comparison between a high-assurance "Full MLS" mode and a high-performance "key management plane."

3.5.4. MLS Library Selection

The MLS Working Group maintains a list of active MLS libraries [28]. The Cisco MLSpp library has been proven to work on a UAS network in previous work [20], so a second library was selected for comparison. The aim is to identify a library less susceptible to the performance issues Marstrander had identified. At the time of selection, OpenMLS was the most recently updated and provided a professional level of documentation and

example code. This would provide a reliable way of creating two high-quality benchmark test harnesses to measure the library's cryptographic performance.

3.5.5. MLS Benchmark Variables

Following an initial familiarisation period to ensure the libraries performed as expected and functionality was fully understood, a test plan was developed. For each implementation, this would test a range of cipher suites across multiple group sizes and measure the time taken to complete four group management operations and three messaging operations. Table 2 shows the test variables.

Table 2. MLS Benchmark Variables.

Library	Ciphersuite	MLS Operation				Messaging
		Add Member	Update Member	Remove Members	Process Update	
OpenMLS	P256_AES128GCM_SHA256_P256 X25519_AES128GCM_SHA256_Ed25519 X25519_CHACHA20_SHA256_Ed25519					100B 1024B 2048B
MLSpp	P256_AES128GCM_SHA256_P256 X25519_AES128GCM_SHA256_Ed25519 P384_AES256GCM_SHA384_P384					

3.5.6. MLS Benchmark Tests

The benchmarks were designed to assess the performance of each library in two *modes*. Full MLS functionality—where MLS is used for key management and message encryption. Key management mode—where the MLS key export function is used to produce symmetric encryption keys for use at the application layer. These modes of operation are shown in Figure 3. This highlights the different positions of the MLS functionality within each architectural stack.

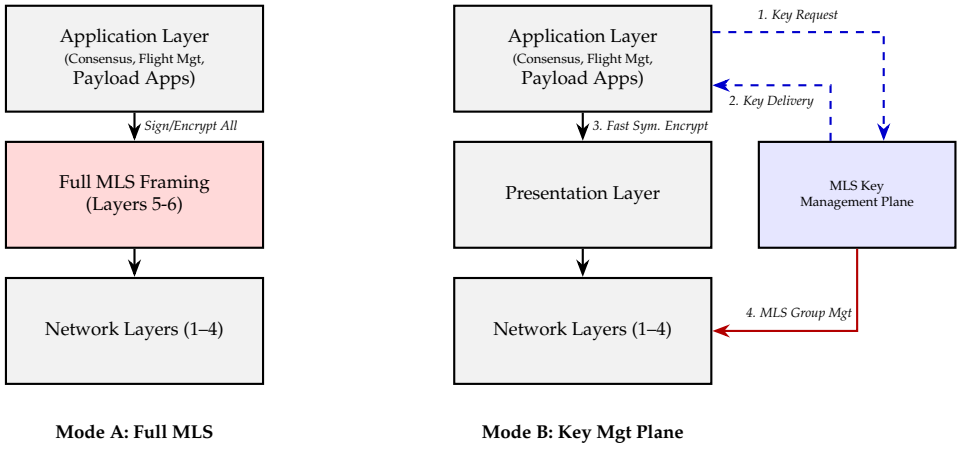

Figure 3. MLS Modes of Operation. Red boxes represent MLS cryptographic processing layers, grey boxes represent application and network layers, blue dashed arrows indicate API calls between the application layer and the MLS key management plane, and black arrows represent the primary data path.

The full MLS functionality benchmarks completed the following:

1. Configure a group (sized 10, 20, 50, 100);
2. Complete an `add member` operation;
3. Complete a `remove member` operation;
4. Complete a `update member` operation;
5. Complete a `process update` operation;

6. Complete a `protect message` and `unprotect message` operation for a message (sizes 100B, 1024B, 2048B).

The key management mode benchmarks completed the following:

1. Process an `export key` request;
2. Use the symmetric key to encrypt a message (sizes 100B, 1024B, 2048B);
3. Use the symmetric key to decrypt a message (sizes 100B, 1024B, 2048B).

The unit of *cost* for these operations is the time taken to complete them. Care was taken for only the time required for the actual operation to be measured. As both libraries implement the same protocol, the benchmarks test how efficiently they implement the protocol. There is no transport layer activity or difference in messaging complexity, so no attempt was made to simulate network activity.

3.5.7. Benchmark Development

All the benchmark tests were run in the previously defined WSL environment and code was written within nano via the WSL command line. AI assistance in the form of Google Gemini was used to iteratively develop the benchmarks and troubleshoot errors. Functional design, validation and analysis were carried out by the researcher. The developed code provided a test harness to manage the input/output of variables into MLS library functions and manage the benchmarking process. A working benchmark was created for the first library and it was used as a template for the second. The benchmark code used during this research and example prompts are published on GitHub (All software components used in this study, including OpenMLS (main branch, accessed on 15 March 2026), MLSpp (main branch, accessed on 15 March 2026), the Criterion benchmarking crate (v0.5.1), and the development environment comprising Microsoft Visual Studio Code (v1.97) with GitHub Copilot and GNU nano within WSL, are publicly available, with the benchmark code and datasets accessible on GitHub (main branch, accessed on 15 March 2026)) [29].

The Rust-based, OpenMLS library is supplied with 'Large_groups.rs' example code. This was used as the foundation of our benchmarks and provided a sound understanding of how the OpenMLS library should be used. The code was then modified to utilise Rust's Criterion crate to apply a more rigorous timing and sample management framework and the send message functionality was created. This was then used as a template for the creation of the MLSpp benchmarks in C++.

In total, four benchmark codes were developed that tested the cryptographic efficiency of two MLS implementations with six cypher suites and groups sized from ten to one hundred. The results showed the impressive scalability and flexibility of the MLS protocol and provided a clear performance comparison between the two libraries. The second half of this research builds on the secure channels established by MLS, considering a different cryptographic problem: proving consensus on a decision within the group.

3.6. Threshold Signature Benchmark Design

In order to verifiably make consensus-based decisions across an autonomous UAS swarm, a method is required to obtain the consent/approval of a threshold number of participating UAS nodes. Threshold signatures provide a mechanism for a node to 'vote for' or securely approve a decision and then for the wider network to only implement that decision if a pre-defined quorum of nodes concur. From the literature review, FROST and BLS signatures appeared best suited to our application:decentralised networks, high group turnover rate, unreliable communications. Elliptic Curve Digital Signature Algorithm (ECDSA) was ruled out due to the complexity of implementing threshold signatures with this algorithm.

Signature performance is driven by curve, protocol and library language. Of these, the chosen protocol drives the message complexity—how many messages the system must send to generate a distributed key or sign a proposal. Table 3 compares each protocol's computation and communication processes.

Table 3. Threshold Signature Rounds.

Round	BLS		FROST	
	Computation	**Communication**	**Computation**	**Communication**
DKG 1	Nodes generate a random polynomial and public commitments.	Each node sends a share to every other node (private).	Each node generates a random polynomial and commits to its coefficients.	Each node privately distributes shares to all other nodes and broadcasts commitments.
DKG 2	Nodes verify their received shares against the public commitments.	Nodes broadcast complaints against any dishonest parties.	Nodes verify their received shares against commitments.	Complaints are broadcast if shares do not match commitments.
DKG 3	Nodes compute the group public key and their final secret key share from all valid shares and commitments.	Disqualified nodes are removed from the group and remaining nodes establish the final group key.	Each participant computes their secret key share (sum of valid received shares).Group public key is derived from commitments.	Misbehaving nodes are excluded; the group agrees on the final group public key.
Sign 1	Each member computes a partial signature using the private key share and a hash of the message.	Partial signatures are sent to a collector or all nodes.	Each signer generates nonces and their corresponding public commitments.	Nonces and commitments are broadcast to all other nodes.
Sign 2	N/A	N/A	Each node creates their partial signature share based on the message and commitments from all nodes.	Signature shares are sent to a collector or broadcast to the network.
Verify 1	The collector or any node computes the final signature by aggregating the partial signatures. A pairing-based equation is checked.	The final signature is broadcast to the network.	A single party checks the validity of the final aggregated signature.The verification is identical to a standard single-party Schnorr signature verification.	The final signature is broadcast to the network.

FROST protocols may use a two-round DKG with a reduced level of security. N/A: Not applicable, indicating that the protocol does not require this step.

3.6.1. Threshold Signature Library Selection

Figure 4 shows the initial candidate libraries and the incremental approach taken to developing these benchmarks to ensure reliable data across all aspects of performance were obtained.

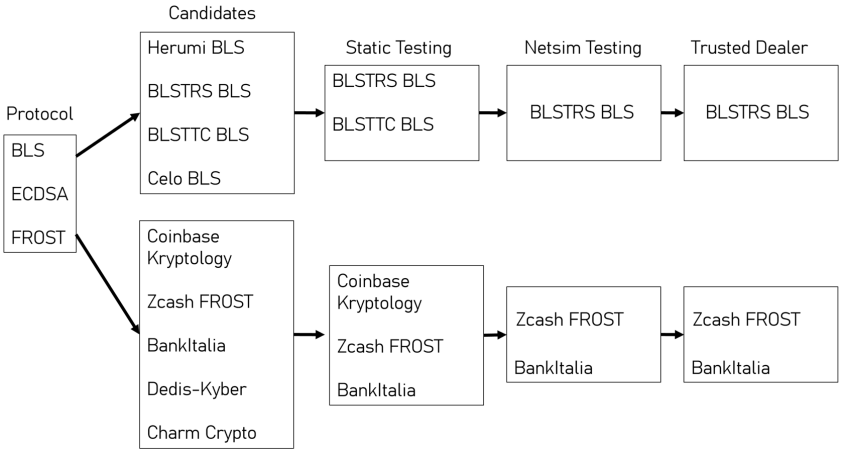

Figure 4. Threshold Signature Benchmark Development.

Starting with static benchmarks that isolated the library's cryptographic functions and ran tests that would allow for a comparison to be made of both the library and the protocol's computational efficiency. Network simulations (netsims) shift from a sequential to a concurrent approach and introduce network latency and loss, providing a realistic simulation of real-world performance. Only the three best-performing libraries were progressed to this stage. This reduced the development burden while maintaining coverage of protocols and cryptographic curves. The benchmarking process was iterative. After conducting the initial single-cycle netsims, the results indicated that the high, one-time cost of DKG was a significant performance bottleneck. To better reflect a realistic use case, where a single key is reused for many signatures, an additional bulk signing benchmark was designed and implemented. Finally, where trusted dealer functionality also existed in the library, the benchmarks were re-run. Allowing for the performance cost of the decentralised key generation to be quantified.

3.6.2. Threshold Signature Benchmark Variables

The variables tested during the static benchmarking process were group size and threshold number. With groups of 5, 10, 20, 40, 60 and 100 all tested with a threshold size of 33 % and 66 %. The group sizes are relevant for a range of tactical and operational use cases and within the scope of the parent project. The thresholds model two scenarios. The first is a highly contested operating environment, where a large number of UASs may be unavailable (due to attrition or signal jamming/obstructions), leading to a requirement for a lower threshold to ensure a quorum. Alternatively, the lower threshold may be deemed sufficient for the type of proposals being approved, in which case the reduced threshold would be expected to improve performance. In the second scenario, a higher threshold is applied, and this could reasonably be expected for decisions that create a risk to the mission (loss of a key capability) or risk to life, trading performance for additional confidence in a proposal's approval. A summary of the tested libraries is in Table 4.

Table 4. Benchmark Summary.

Source Library	Language	Curve	Key Generation	Timing	Comm Method
Kryptology	Go	Ed25519	DKG/TD	testing.B	Buffered Channels
Kryptology	Go	Secp256k1	DKG	testing.B	Buffered Channels
Zcash Foundation	Rust	Ed25519	DKG	Criterion	BTreeMap
Bank Italia	C++	secp256k1	DKG/TD	Google Benchmark	Arrays
Filecoin-BLSTRS	Rust	BLS12-381	DKG	Criterion	BTreeMap
BLSTTC	Rust	BLS12-381	DKG	Google Benchmark	BTreeMap

3.6.3. Threshold Signature Benchmark Tests

For each library benchmarked, the following operations would be tested:

1. **DKG** Measuring time to generate a group key and provide each participant with their share of the secret key. Expected to be the most resource-intensive operation and required during swarm configuration or when adding a new node.
2. **Signing** Measuring time to generate a threshold signature. Expected to be the second most demanding operation; keys can be reused to sign multiple proposals, so this is expected to be the dominant operation.
3. **Signature verification** Measuring time to verify a signature. Frequency will match signing, with FROST and BLS having different verification processes and resource demands.
4. **End-to-end operation** Complete protocol timing for all phases, providing a comprehensive metric for comparison.

3.7. Network Simulations

To capture the effect of communication overheads, benchmarks incorporating simulated network conditions were developed. Artificial delays were introduced to emulate message latency, and randomised message drops were used to represent packet loss. The implementation of this simulation was language-specific: in Rust, delays and drops were introduced directly at the benchmark harness level; in C++ and Go, equivalent logic was implemented using the respective concurrency and timing libraries. This ensured consistency in the simulated conditions across all tested libraries, while allowing for each benchmark to be executed natively within its language environment. The network scenarios vary the latency and packet loss with a normal distribution as shown in Table 5.

Table 5. Network Performance Tests.

Source Library	Language	Curve	Key Generation	Network Parameters	Group Sizes
Zcash FROST	Rust	Ed25519	DKG/TD	50 ± 10 ms/120 ± 30 ms, 2%/5% packet loss	5, 10, 20, 30, 40
Filecoin-BLSTRS BLS	Rust	BLS12-381	DKG	50 ± 10 ms/120 ± 30 ms, 2%/5% packet loss	5, 10, 20, 30, 40
BankItalia FROST	C++	secp256k1	DKG	50 ± 10 ms/120 ± 30 ms, 2%/5% packet loss	5, 10, 20, 30, 40

3.8. Trusted Dealer

For comparison against decentralised Distributed Key Generation (DKG), benchmarks were also conducted using a trusted dealer (TD) model. In these tests, a single client acted as the dealer, distributing secret key shares to each participant. This simplified protocol incurs only one round of communication latency and packet loss (dealer to participant), avoiding the multiple broadcast and verification stages of DKG. Including TD benchmarks provides an equivalent reference point for computation and communication costs, allowing for the overhead of decentralisation to be quantified.

Benchmark Environment

Microsoft Visual Studio Code with integrated GitHub AI Copilot was used for code development. This allowed for rapid prototyping of 11 benchmarks in three different programming languages. The code provided a test harness to manage the input/output of variables into the FROST/BLS library functions, group participant data and orchestrate the signing operations and generate statistical data. A working benchmark was created for the first library and it was then used as a template for subsequent prompts, all code and prompts are published on GitHub [29] (All software components used in this study, including OpenMLS (main branch, accessed on March 2026), MLSpp (main branch, accessed on 15 March 2026), the Criterion benchmarking crate (v0.5.1), and the development environment

comprising Microsoft Visual Studio Code (v1.97) with GitHub Copilot and GNU nano within WSL, are publicly available, with the benchmark code and datasets accessible on GitHub (main branch, accessed on March 2026)).

When developing and testing the code, care was taken to ensure consistent performance measurements were being taken, with the Coefficient of Variation data used to measure stability of the benchmarking process. Time measurements for individual phases of the signing process were recorded in order to allow for identification of bottlenecks in the signing process, which may be incompatible with the real-world concept of employment for the UAS network. To ensure similar levels of precision timing, Criterion (Rust), Google Benchmark (C++) and testing.B (Go) frameworks were used to manage iteration/sample sizes, timing and collation of results for each operation. The exception to this was the netsim benchmarks, where these formal benchmarking frameworks proved too time-consuming (in excess of 24 h) for the largest groups/worst network conditions. As an alternative, a fixed number of 10–20 iterations was performed for each data point. This pragmatic trade-off allowed for the collection of essential performance data in realistic scenarios, while acknowledging a reduced level of statistical precision compared to the static benchmarks.

A language-appropriate method of passing messages between the simulated group participants was used (Channels, Maps and Arrays). For the static benchmarks, this did not attempt to introduce variations in network performance as a variable, but provided a method for simulated clients to pass messages in an ordered manner. Supporting DKG for both protocols and signing for the FROST benchmarks (with BLS only having a single signing round, this was not required). Network performance was isolated and tested separately.

4. Results and Discussion

This project aimed to comprehensively assess the performance of MLS and Threshold Signature libraries and, in doing so, identify suitable implementations for use within a decentralised network of UAS. Table 6 summarises the benchmarks that were completed. Combined, these provide an extensive dataset from which to produce insights on the performance impacts of cryptographic curve, protocol and language implementation.

Table 6. Completed Benchmark Summary.

Protocol	Library	Description
MLS	OpenMLS	Key distribution & message encryption modes
MLS	MLSpp	Key distribution & message encryption modes
FROST	Kryptology	Ed25519 DKG (static)
FROST	Kryptology	Secp256k1 DKG (static)
FROST	Zcash	Ed25519 DKG (static), bulk signing & TD
FROST	BankItalia	Secp256k1 DKG (static), netsim, bulk signing & TD
BLS	BLSTTC	BLS12-381 DKG (static)
BLS	BLSTRS	BLS12-381 DKG (static), netsim, bulk signing & TD

4.1. MLS Results

The benchmark results showed a clear performance advantage for the OpenMLS library compared to the MLSpp library. Full results for all the cipher suites tested can be found in Appendix A. The following sections will consider the performance of each library in turn, highlighting key results and providing supporting data and analysis of their implications.

4.2. OpenMLS Results

4.2.1. Full MLS Mode Benchmark

OpenMLS showed consistently high levels of performance across all cipher suites. Increasing the client number from 10 to 100 had a relatively small impact on performance, with the `add member` operations increasing by a factor of 1.5× to 2.4× as the group size increased from 10 to 100. `Send message` performance showed no measurable difference as group size increased and only a slight increase as message size increased (Table 7).

Table 7. OpenMLS Full Results.

Ciphersuite	Group Size	Add Member	Update Member	Remove Member	Process Update	Msg (100B)	Msg (1024B)	Msg (2048B)
P256_AES128GCM_SHA256_P256	10	5.14	3.16	2.11	2.17	0.630	0.675	0.732
	20	5.32	3.40	3.37	3.21	0.712	0.690	0.704
	50	6.06	5.60	5.13	3.77	0.667	0.682	0.645
	100	8.11	6.58	6.75	4.66	0.700	0.757	0.756
X25519_AES128GCM_SHA256_Ed25519	10	2.01	0.96	0.77	0.84	0.117	0.122	0.136
	20	2.15	1.88	1.19	1.32	0.122	0.142	0.136
	50	3.07	2.50	2.43	2.09	0.163	0.164	0.152
	100	4.44	3.81	3.66	2.91	0.178	0.175	0.193
X25519_CHACHA20_SHA256_Ed25519	10	1.81	1.28	0.95	0.80	0.126	0.133	0.146
	20	2.47	1.82	1.18	1.11	0.131	0.159	0.150
	50	3.17	2.49	2.12	2.01	0.143	0.173	0.163
	100	4.41	3.69	3.52	3.02	0.172	0.194	0.209

4.2.2. MLS Key Management Mode Benchmark

The combined results for a single Key Export–Encrypt–Decrypt cycle, using the X25519 AES 128 SHA256 Ed25519 cipher suite (also used for equivalent MLSpp benchmark), are shown in Table 8. This shows performance reducing as both message size and group size increase. Key export times dominate these results, for 100-client groups a performance reduction between 2.3× and 3.5× was observed relative to the 10-client group. A smaller reduction in performance was seen as message size increased from 100B to 2048B, between 1.2× and 1.8×.

Table 8. OpenMLS Key Management Mode Results.

Group Size	100B Median (ms)	1024B Median (ms)	2048B Median (ms)
10	0.0041	0.0060	0.0073
20	0.0057	0.0070	0.0086
50	0.0089	0.0108	0.0121
100	0.0144	0.0155	0.0170

4.3. MLSpp Results

4.3.1. Full MLS Mode Benchmark Results

MLSpp showed lower levels of performance, with far more variation across all operations. `add member` operations scaled from 1 to 3× as group sized increased from 10 to 100. `Update Member/Remove Member/Process Update` showed even larger increases, up to 8.3× in the worst case (X25519—`Remove Member`—10 to 100 clients). This is a particularly significant finding, as it shows that a core group management function scales linearly, failing to achieve the theoretical logarithmic scaling of the MLS protocol. Send message performance remained consistent as group size increased, but showed an average increase of 1.6× as message size increased from 100B to 2048B (and 2.2× in the worst case.). These results are shown in Table 9.

Table 9. MLSpp Full Mode Benchmark Results.

Ciphersuite	Group Size	Add Member (ms)	Update (Create) (ms)	Update (Process) (ms)	Remove Member (ms)	Msg (100B) (ms)	Msg (1024B) (ms)	Msg (2048B) (ms)
P256_AES128GCM_ SHA256_P256	10	4.72	8.19	3.86	8.13	0.830	1.267	1.463
	20	5.61	15.00	5.30	14.06	0.818	1.085	1.507
	50	8.10	33.23	8.26	35.01	0.802	1.261	1.534
	100	14.31	56.12	11.36	58.71	0.777	1.108	1.314
X25519_AES128GCM_ SHA256_Ed25519	10	3.65	3.75	2.35	3.22	0.614	0.989	1.192
	20	3.61	5.61	2.96	5.09	0.560	0.949	1.205
	50	6.35	12.13	5.18	21.23	0.777	0.864	1.229
	100	9.82	23.58	9.14	26.92	0.582	0.939	1.309
P384_AES256GCM_ SHA384_P384	10	37.05	57.09	18.29	62.99	5.982	5.881	5.944
	20	35.59	95.87	20.35	105.04	5.400	18.079	6.127
	50	33.08	213.44	25.74	212.01	5.637	5.924	6.405
	100	37.41	431.62	32.04	400.91	4.855	5.040	5.703

4.3.2. Key Management Mode Benchmark Results

In this mode, MLSpp showed far less variation across all operations and end-to-end performance. The key export performance showed no measurable difference as the group size increased. Encryption and decryption performance showed a 2.1× and 1.9× increase in time as message size increased from 100B to 2048B. Again key export times dominated the end-to-end results. With OpenMLS performing twice as well as MLSpp for the 100B message across a 10 person group, and the results reversed for the largest group size, where a 1.65× advantage was observed for MLSpp. Table 10 shows the end-to-end results for MLSpp.

Table 10. MLSPP Key Management Mode Benchmark Results.

Group Size	100B Median (ms)	1024B Median (ms)	2048B Median (ms)
10	0.0083	0.0094	0.0104
20	0.0077	0.0093	0.0106
50	0.0079	0.0093	0.0103
100	0.0077	0.0089	0.0103

4.3.3. Open MLS vs. MLSpp

OpenMLS comprehensively outperforms MLSpp for all aspects of the MLS Full Mode benchmark. Figure 5 visualises this result. It is noteworthy that MLSpp's update and remove member performance scale so poorly. While these operations involve modifying data stored within the secret tree, they are less cryptographically demanding than the add member operation. This suggests that the library is less well optimised for this data management task, particularly for larger group sizes. In comparison, OpenMLS achieved close to the theoretical $O(\log n)$) performance limit for all operations. This result corroborates the findings of Marstrander [20], that MLSpp suffered data marshalling issues and was a sub-optimal choice for this type of application. Under the active node compromise and churn assumptions defined in Section 3.4, dynamic group operations become the dominant cost factor. In this context, OpenMLS significantly outperforms MLSpp, whose poorly scaling group management functions introduce rekey delays that would exceed tolerable recovery times following node compromise.

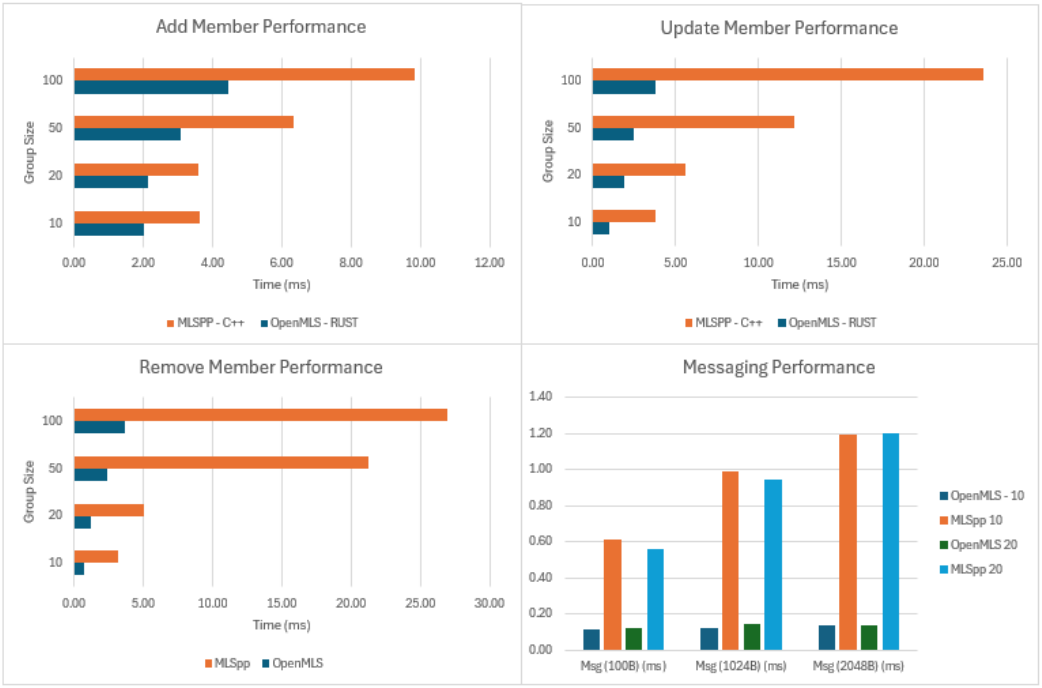

Figure 5. OpenMLS vs. MLSpp Performance.

4.3.4. Full vs. Key Management Mode

The key export function allows for the production of a symmetric session key that can be used by a separate application for encryption. This results in far higher throughput rates than using the `message protect` and `message unprotect` operations in 'Full MLS Mode'. Messages are no longer signed, so there is a reduction in assurance, there is also a corresponding reduction in demand on the system. So it is unsurprising that performance is significantly greater in Key Management mode, this difference is shown in Figure 6, where the time taken for OpenMLS to complete one encryption cycle is 20× greater for a 1024B message and a group size of 10.

While these benchmarks were conducted on an x86-based 12th Gen Intel i5 processor to provide a high-fidelity computational baseline, it is acknowledged that absolute execution timings will scale higher on the embedded ARM-based architectures typical of UAS flight controllers. However, the 20× performance delta observed between 'Full MLS' and 'key management' modes is fundamentally driven by the reduction in cryptographic signing operations rather than hardware-specific optimisations. While further research is required to quantify the exact scaling factor on hardware lacking AES-NI acceleration, the relative gain represents a mandatory reclamation of 'computational headroom' for any resource-constrained platform operating in contested environments. By isolating these algorithmic efficiencies on a standardized platform, we provide a reproducible metric for the minimum performance overhead developers must account for. Within the assumed DoS and intermittent connectivity threat model (Section 3.4), this reclaimed computational headroom directly improves survivability by reducing the window in which a swarm operates with stale or partially compromised cryptographic state

Thus, the identification of this high-efficiency mode remains the most significant finding for developers seeking to implement resilient, secure autonomy on resource-constrained platforms. Both libraries showed similar stand alone encryption/decryption performance (within one microsecond of each other), although MLSpp showed a 7.5 microsecond advantage over OpenMLS when exporting a key. However, this small advantage becomes irrelevant if a key is used more than eight times (in the worse case scenario) due to OpenMLS' superior encryption/decryption performance.

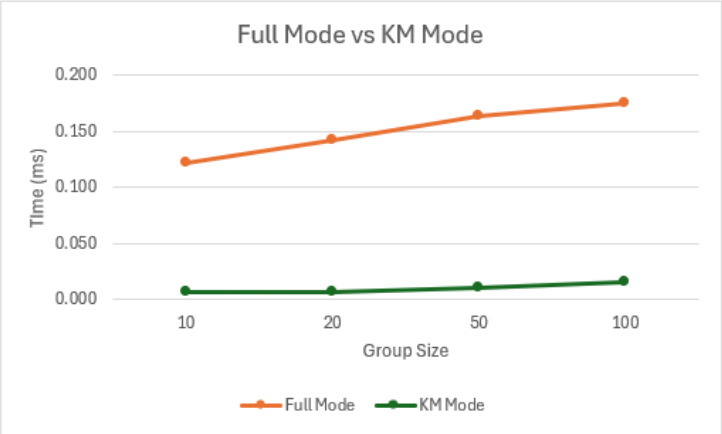

Figure 6. Full mode vs. Key Management Mode.

There are benefits beyond just increased throughput. Messages sent using the `Message Protect` and `Message Unprotect` operation are dependent on the Delivery Service (DS) to correctly order messages as they are distributed to the group, if the DS is temporarily unavailable messages cannot be sent. The DS also has to be available for handshake messages to be sent, however the frequency with which these occur is controlled by the system integrating MLS and a DS outage is tolerable. The existing key will still work and when the DS is restored, the system will rekey, with any compromises that have occurred being healed. The practical limit of this tolerance would be an appropriate area for further study. A second additional benefit is that removing encryption/decryption duties reduces the processing load on the system. For UAS systems with limited processing and battery power, this is an important consideration. No attempt was made to resource constrain these benchmarks to model representative hardware, but it is likely that MLS as a key management plane has greater utility on these kinds of systems and this should also be a focus of further study.

The trade-off for increased performance in this case is a loss of assurance from the application of digital signatures to every message. It can be argued that this is useful in a messaging application (Wickr/Signal), where attribution of who sent a message is vital to the integrity of the group conversation. However that has less utility in a machine to machine exchange where 1000's of messages are being sent and the cost overhead quickly becomes unsustainable. The use of MLS in this way is recognised in the RFC and is not a subversion of the protocol.

The benchmarks reveal that the computational demand of 'Full MLS Mode' is unsustainable for autonomous UAS operations. In machine-to-machine exchanges where thousands of messages are processed per minute, the security benefit of per-message digital signatures is negligible. However, the 20× performance penalty observed in our tests represents a significant drain on the onboard 'computational headroom'. In contested environments where the processor must handle unpredictable spikes in demand due to adversary attacks and maintaining flight-stability logic, this 20x saving is critical for system resilience. Consequently, we recommend the use of MLS exclusively as a key management plane to provide Post-Compromise Security (PCS) while preserving CPU cycles for mission-related tasks.

4.3.5. Impact of Cipher Suite

While library selection has a much greater impact on performance, the choice of cipher suite also has an effect. In Figure 7, the top three graphs show that the time taken to complete an `add member` increased with group size. The slowest suite (top line)

utilised the NIST P256 signature algorithm, which is used multiple times during this operation, with a similar impact on the send message performance. In comparison, the two curves that utilise the faster Ed25519 signature scheme have near identical performance, despite their use of different symmetric encryption keys. Previous work [19,20] utilised the suites containing P256 due to that algorithm being certified by NIST for use with government contracts. This is an understandable approach, and customer requirements may dictate future specifications. However, the forecast arrival of quantum computers will undermine these classical algorithms security. Neither library fully supports a post-quantum secure cipher suite but integration of ML-KEM into OpenMLS is imminent [30]. However, no digital signature solution is yet available. In the meantime, focussing customer requirements on selecting enduring key exchange and encryption algorithms in order to maintain confidentiality is likely to be a higher priority than the use of a NIST approved digital signature. The likely increase in size and processing requirements for PQC secure signatures, will further reduce the performance of the 'full MLS mode' within our use case and strengthens the argument in for adopting it as a key management plane.

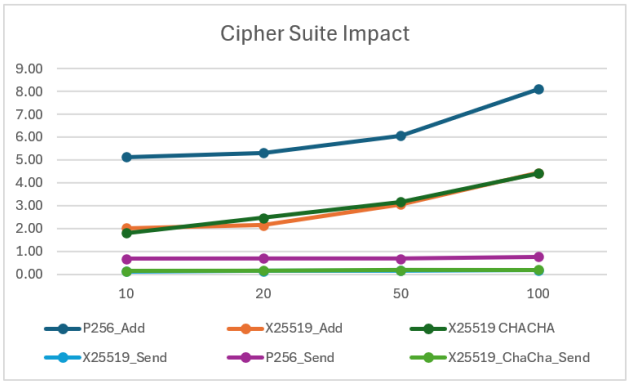

Figure 7. OpenMLS Cipher Suite Performance.

4.3.6. MLS Results Conclusion

The chosen metrics and specific MLS operations benchmarked provided sufficient insight into the protocols operation and the libraries performance for an in depth assessment of their suitability to be made. With OpenMLS shown to be the highest performing library across all benchmarks and the performance issues identified with MLSpp corroborated by our test. The use of MLS in key management mode was shown to be significantly higher performing at encryption/decryption than in full mode, and this mode is likely to have benefits for low-power/-compute networks like our UAS concept beyond just speed, although these remain unproven and would be a key test for future extensions of this work.

While the primary MLS group management operations were tested, the group setup phase was not. Having been deliberately excluded from the benchmark, following the OpenMLS example benchmark. This configures the group states as a separate operation outside of the benchmarking process. It would also be completed during the configuration phase within our proposed concept of operation, so timeliness is important but not critical.

4.4. Threshold Signature Benchmarks

Full, raw results for each of the benchmarks discussed can be found in Appendix B. More focussed snapshots are provided within this section where it is appropriate.

4.4.1. Static Results

The FROST implementations utilising the Ed25519 curve were expected to provide the highest performance, assuming similar levels of optimisation were achieved within

the developers' implementation. Table 11 shows the aggregated end-to-end results for all benchmarks, with 'end-to-end' measuring a single DKG, Sign and Verify cycle. This shows a worse drop in performance than the theoretical forecast of O(n2) growth as the number of nodes expands. With the best performing library experiencing a 3000× reduction in performance (Bank of Italia) as the number of nodes increased from 5 to 100 and the worst suffering a 5000× reduction (BLSTRS). This significant drop in performance confirms that the DKG process is the primary bottleneck in the system. However, a key finding from these results is that the fastest implementation used a Secp256k1 curve, challenging the hypothesis that curve selection would be a key factor in performance. The Kryptology library was tested with two curves, Ed25519 being the default and Secp256k1 as an alternative. The results show a 600% difference in performance for a given group size/threshold. Suggesting that library-specific optimisation for a curves and the quality of the overall implementation have a bigger impact on performance than using a fast or slow curve.

When the performance of individual operations is examined, the impact of the protocols messaging complexity becomes more pronounced. When run statically, without any network impact, BLS single-round signing protocol results in a faster signing time than FROST's multi-round process, which is a key trade off between the protocols. This is shown by the results in Figure 8.

The results also highlight the highly optimised nature of the different library implementations. For instance, the Zcash library demonstrates exceptionally fast signing performance (28.91 ms for 100 nodes), outperforming the "end-to-end" fastest library, Bank Italia (291.07 ms), by an order of magnitude in this specific phase. However, its DKG performance (24,357 ms) is more in line with other the Kryptology-Ed25519 library (22,342 ms). This suggests a specialised optimisation for signing operations within the Zcash library, making it a strong candidate for use cases where signing frequency is high. Verification is shown to be constant regardless of group size, which is as expected as it only involves the group public key and the signature. With the more computationally demanding bilinear pairing operation of BLS requiring more resource than FROST, shown in Figure 9.

Table 11. Results—Static End-to-End Median Time (ms).

Threshold	Nodes	FROST-DKG				BLS-DKG
		Kryptology Ed25519	**Kryptology Secp256k1**	**Zcash Ed25519**	**BankItalia Secp256k1**	**BLSTRS BN12**
33%	5	7.87	50.63	6.65	4.37	8.47
	10	54.34	263.06	39.09	23.80	48.34
	20	283.42	1436.12	236.09	134.00	303.97
	40	1947.57	11,697.41	1708.33	922.00	2370.42
	60	5852.95	41,344.78	5446.51	2834.00	7646.57
	100	26,535.31	173,948.96	24,865.36	12,815.00	35,705.35
66%	5	11.56	79.17	10.51	7.38	14.99
	10	68.72	427.51	60.46	36.20	82.58
	20	392.35	2724.29	433.71	239.00	604.81
	40	276.89	19,326.77	3205.45	1761.00	4522.32
	60	12,114.81	80,340.13	10,477.19	5472.00	15,297.59
	100	54,228.44	347,080.33	47,916.47	27,871.00	70,334.35

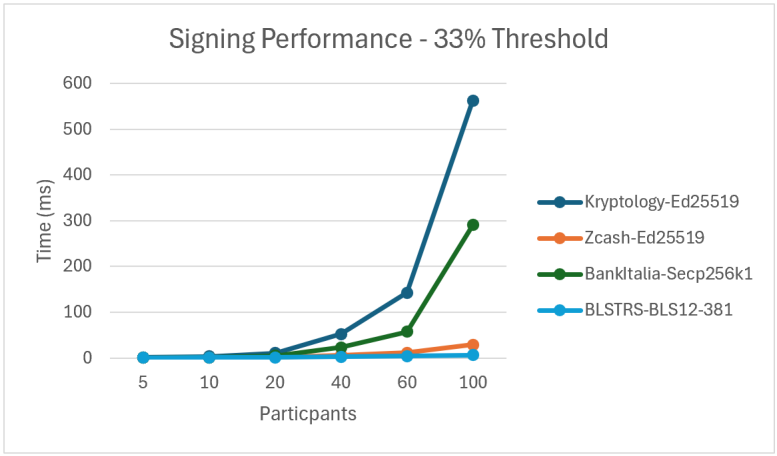

Figure 8. Results—Signing Performance.

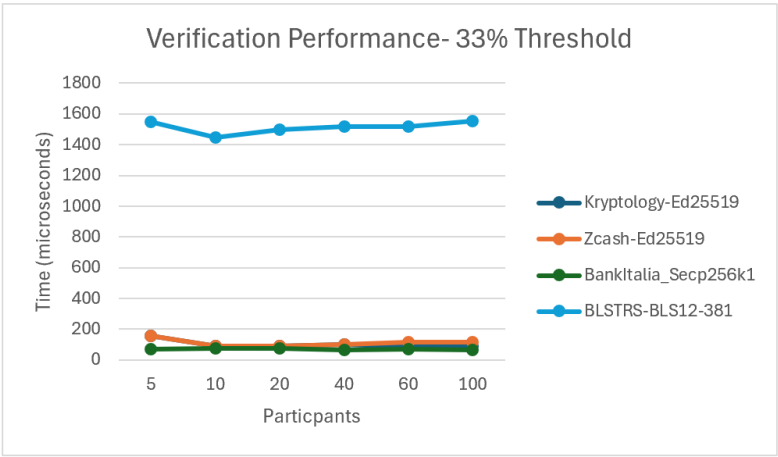

Figure 9. Results—Verification Performance.

Finally, while the benchmarks measure a single DKG, sign, verify cycle. The most likely use case is for a key to be reused to sign multiple proposals—keys only need to change when a new member is added to the group or a key compromise is suspected. To highlight the impact this has on comparative performance, Figure 10 shows the static cost of 1 DKG cycle with 20,000 sign–verify cycles (calculated rather than simulated). In this situation, the advantage of faster sign–verify times from the Zcash and BLSTRS implementations becomes clear. Showing that while DKG time is a critical configuration bottleneck, the efficiency of the signing and verification phases is the primary factor for enduring operation.

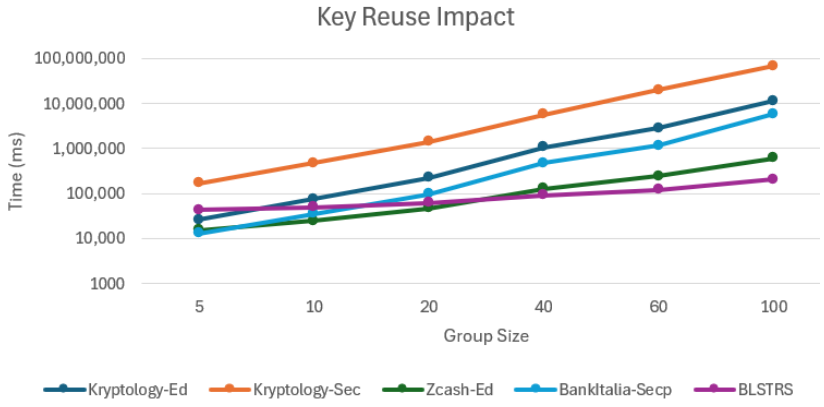

Figure 10. Results—Key Reuse Impact.

4.4.2. Network Simulation Results

While the static benchmarks measure the computational efficiency of the library by sequentially measuring every operation. They do not provide a real-world estimation of performance. To fill this gap we developed netsims that fixed the group threshold and then varied network performance for a given group size. Example results for the BLSTRS BLS implementation are in Table 12. While the AI copilot facilitated rapid prototyping of static benchmarks, the implementation of asynchronous retry logic in the network simulations (netsims) revealed a significant prompt dependency. The resulting variability in AI-generated approaches necessitated a rigorous manual verification phase, leading to the prioritisation of two validated benchmarks to ensure the integrity of the reported performance metrics. Despite this, the impact of both network performance and protocol design become apparent. With more communication rounds comes more opportunity for delays and reduced end-to-end performance. With the FROST netsim degrading by 76% for a 40 node group in the worst case network scenario, in comparison the BLS netsim only degraded by 10% for the same group size. However, this is over a single DKG sign–verify cycle. As such a decision was made to focus remaining time on developing an bulk signing netsim.

Table 12. Results—BLSTRS BLS Netsim.

50 ms ± 10 ms + 2% Packet Loss									
40% Threshold		DKG		Signing		Verification		End to End	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	231.28	29.6	75.35	40.7	3.38	28.1	310.23	16.7
10	4	322.57	113.2	68.15	41.6	3.05	45.5	398.73	16.6
20	8	931.26	21.8	79.59	37.6	2.12	64.6	1050	46.5
40	16	5150	5	156.92	28.3	3.14	32.4	5290	4.3
120 ms ± 30 ms + 5% Packet Loss									
40% Threshold		DKG		Signing		Verification		End to End	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	381.53	11.5	157.11	44.7	138.5	47.9	534.21	16.9
10	4	552.88	14.7	177.25	40.1	125	36.1	755.79	13.2
20	8	1200	9.7	177.36	16.7	108.5	27.1	1400	8.5
40	16	5580	8.3	225.01	13.7	137	28.1	5810	7.4

4.5. Trusted Dealer Comparison

Figure 11 highlights the stark performance gap between DKG and Trusted Dealer (TD) setups in Zcash. While TD setup times remain almost flat across all group sizes, DKG scales relatively poorly, reaching over 31 s at 100 nodes—nearly 200× slower. This highlights the significant cost of decentralisation—similar results were observed for all libraries; full results are in Appendix B.

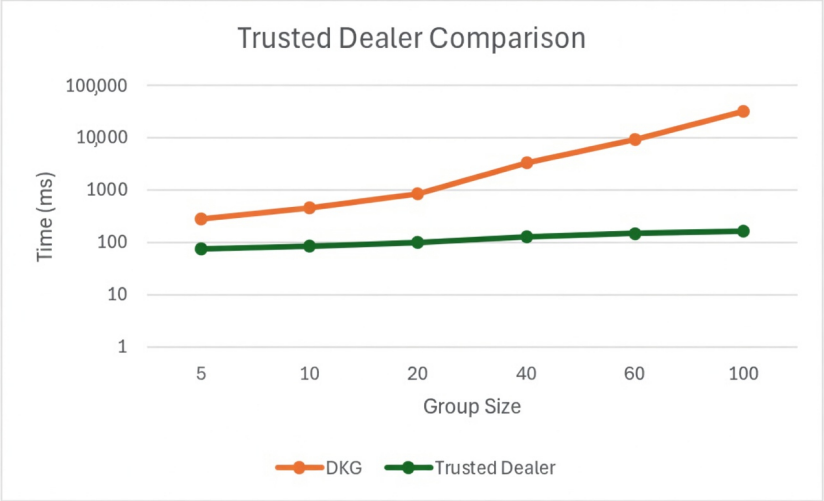

Figure 11. Results—Zcash Trusted Dealer Comparison.

4.5.1. Bulk Signing Netsim

We have shown that DKG is the primary bottleneck in both BLS and FROST signing processes. We have highlighted the difference in sign/verify performance between libraries and protocols, and we have shown that the FROST netsim was less resilient to degraded network performance (over a single signing cycle). In real life, a DKG round would generate a key used for 100 s or 1000 s of signatures, during these 1000 s of signatures, multiple packets will be dropped and have to be retried. To assess the impact of spreading the DKG 'cost' over multiple signatures bulk signing netsims were created. These would test a key being used for a range of signature cycles (10, 100, 200, 400), for a fixed threshold (40%) and network performance (50 ms +/−10 ms +2% packet loss), retry logic was fixed at three attempts with linear back off to ensure a consistent methodology across the languages and libraries.

These netisms show that the Zcash implementation, which was seen to be highly efficient for sign and verification operations during static testing, was the best performing library during the bulk signing tests, consistently 20 ms faster than the second placed library (BLSTRS). The results also show that spreading the DKG costs has rapidly diminishing returns, with the cost effectively paid off after the first 100 signatures (Figure 12).

Figure 12. Results—Per-Signature Performance.

Figure 13 shows the impact increasing group size has on performance, for the Zcash library's 400 cycle test, doubling the group size from 20 to 40 only increased the per-

signature costs by ∼15% with a similar result observed for the BLSTRS library (∼20%). This is a critical result, it shows that with a reasonable volume of key reuse, even large groups can maintain an acceptable throughput. The tests were only for a single network performance scenario and were only scaled up to a group size of 40. Future work should further investigate both of these variable's impacts.

Figure 13. Results—Impact of Group Size.

4.5.2. Threshold Signature Results Conclusion

These benchmarks have shown the validity of our methodology; the static benchmarks allowed for the raw efficiency of each library to be assessed. Meanwhile, the netsims placed them in a realistic simulation that allowed for both protocol design and library efficiency to be evaluated. DKG was a clear constraint on performance for all libraries. However the Zcash FROST library's faster DKG and verification causing it to outperform BLSTRS. The variation in performance stems not just from library optimisation, but also from the differences in BLS and FROST communication and computation protocols shown in Section 3.2. FROST uses an interactive, multi-round signing process to provide stronger real-time security guarantees. This prevents a malicious actor from manipulating their signature share partway through the signing process and allows for the group to exclude the actor from future rounds, at the cost of an extra round of communication. In the single-cycle netsim, performance was degraded 76% in the worst case scenario. In comparison, BLS's non interactive signing phase is faster and more resilient to network degradation, as shown in the netsim. However it places a heavier responsibility on the initial DKG phase to establish a secure group. It also requires a much more costly verification step, which combined with the network time required to collect partial signature becomes a significant factor in high throughput scenarios. Arguably there is no 'best' signing algorithm, however in these benchmarks the Zcash implementation was shown to be the best performing for our use case. It is also from a credible developer and appears well maintained, an important consideration for its integration into our wider project.

4.6. Decision Framework

While 'Full MLS' mode provides per-packet non-repudiation, the 20× computational overhead is prohibitive for latency-vulnerable data streams such as Full Motion Video. Consequently, the 'key management' mode is identified as the optimal architecture for the UAS data plane. Table 13 synthesises these observed trade-offs into a selection framework for system developers.

Table 13. UAS Cryptographic Selection Framework based on Mission Constraints.

Priority	Optimised Stack	Measured Benefit	Operational Context
Maximum Assurance	OpenMLS (Full) + FROST	Per-packet attribution	High-stakes control-plane commands requiring non-repudiation.
High Throughput	OpenMLS (KM) + FROST	20× speed reclamation	Latency-critical FMV and telemetry streams where signing is prohibitive.
Network Resilience	OpenMLS (KM) + BLS	7× lower re-key latency	Contested RF environments with high packet loss and disrupted links.
Scalability	OpenMLS (KM) + FROST	$O(\log n)$ efficiency	Large-scale autonomous coordination for swarms exceeding 60 nodes.

5. Conclusions

The results presented in the previous section quantify the performance and scaling behaviour of the evaluated cryptographic mechanisms; this section interprets those findings in the context of autonomous UAS design, with particular emphasis on resilience, scalability, and operational feasibility under contested conditions.

The challenge of securing decentralised autonomous systems, such as swarms of UAS, is a critical barrier to their widespread adoption. To be effective, these systems must operate without access to centralised trust authorities and in contested environments, requiring cryptographic solutions that are not only secure but also high-performing. This paper seeks to address this challenge by conducting a comprehensive performance evaluation of modern cryptographic libraries for two key functions: secure group communication using Messaging Layer Security (MLS) and decentralised consensus using threshold signatures (FROST and BLS). Importantly, all performance conclusions in this work should be interpreted relative to the threat model defined in Section 3.4, which prioritises resilience to node compromise, denial of service, and high membership churn, and therefore places particular emphasis on rekey latency, recovery behaviour, and sustained performance under non-ideal network conditions. The extensive benchmarking conducted in this research provided a clear, evidence-based assessment of the available technologies. For secure group communication, the research demonstrated that the OpenMLS library is a high-performing and scalable solution, far exceeding the capabilities of MLSpp, particularly in the group management operations critical for dynamic UAS swarms. Furthermore, the findings showed that using MLS in a *key management* mode offers a dramatic increase in performance and resilience, making it a highly suitable choice for resource-constrained UAS networks.

For decentralised consensus, the benchmarks revealed a range of compromises to consider. The Zcash FROST implementation emerged as the most suitable all-round performer for high-volume, realistic use cases, balancing performance with strong security guarantees. A key finding was that while DKG is the primary setup bottleneck, the recurring cost of signing, verification and network communication in sustained operations is a more critical factor in overall throughput. Finally, the results consistently showed that practical performance is dictated not just by the theoretical protocol, but by the quality of the library implementation.

Future work will extend this evaluation to more tightly constrained execution environments and to the integration of cryptographic mechanisms within complete autonomous

UAS software stacks. In particular, assessing performance on representative embedded and avionics-class hardware, and examining behaviour under sustained operation with dynamic group membership and network disruption, would further characterise the practical limits of these approaches. Such evaluation would complement the results presented here by providing additional insight into long-duration operation and deployment-level considerations.

Author Contributions: Conceptualization, P.R. and W.J.B.; methodology, P.R. and W.J.B.; software, P.R. and W.J.B.; validation, P.R., W.J.B. and R.M.; formal analysis, P.R. and W.J.B.; investigation, P.R. and W.J.B.; writing—original draft preparation, P.R. and W.J.B.; writing—review and editing, P.R., W.J.B., R.M. and M.T.; supervision, W.J.B. and R.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: We have made the code and data available on the related GitHub https://github.com/RochP-Lab/Dissertation (accessed on 20 August 2025).

Acknowledgments: The authors acknowledge the support of the Blockpass ID Lab in the development of this work.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. MLS Results

Table A1. MLSpp Full MLS Mode Results.

Ciphersuite	Group Size	Add Member	Update (Create)	Update (Process)	Remove Member	Msg (100B)	Msg (1024B)	Msg (2048B)
P256_AES128GCM_SHA256_P256	10	4.72	8.19	3.86	8.13	0.830	1.267	1.463
	20	5.61	15.00	5.30	14.06	0.818	1.085	1.507
	50	8.10	33.23	8.26	35.01	0.802	1.261	1.534
	100	14.31	56.12	11.36	58.71	0.777	1.108	1.314
X25519_AES128GCM_SHA256_Ed25519	10	3.65	3.75	2.35	3.22	0.614	0.989	1.192
	20	3.61	5.61	2.96	5.09	0.560	0.949	1.205
	50	6.35	12.13	5.18	21.23	0.777	0.864	1.229
	100	9.82	23.58	9.14	26.92	0.582	0.939	1.309
P384_AES256GCM_SHA384_P384	10	37.05	57.09	18.29	62.99	5.982	5.881	5.944
	20	35.59	95.87	20.35	105.04	5.400	18.079	6.127
	50	33.32	213.44	25.74	212.01	5.637	5.924	6.405
	100	37.41	431.62	32.04	400.91	4.855	5.040	5.703

Table A2. OpenMLS Full Mode Results.

Ciphersuite	Group Size	Add Member	Update Member	Remove Member	Process Update	Msg (100B)	Msg (1024B)	Msg (2048B)
P256_AES128GCM_SHA256_P256	10	5.14	3.16	2.11	2.17	0.630	0.675	0.732
	20	5.32	3.40	3.37	3.21	0.712	0.690	0.704
	50	6.06	5.60	5.13	3.77	0.667	0.682	0.645
	100	8.11	6.58	6.75	4.66	0.700	0.757	0.756
X25519_AES128GCM_SHA256_Ed25519	10	2.01	0.96	0.77	0.84	0.117	0.122	0.136
	20	2.15	1.88	1.19	1.32	0.122	0.142	0.136
	50	3.07	2.50	2.43	2.09	0.163	0.164	0.152
	100	4.44	3.81	3.66	2.91	0.178	0.175	0.193
X25519_CHACHA20_SHA256_Ed25519	10	1.81	1.28	0.95	0.80	0.126	0.133	0.146
	20	2.47	1.82	1.18	1.11	0.131	0.159	0.150
	50	3.17	2.49	2.12	2.01	0.143	0.173	0.163
	100	4.41	3.69	3.52	3.02	0.172	0.194	0.209

Table A3. Key Export, Encryption/Decryption, and Total Time Performance.

OpenMLS			**MLSpp**		
Key Export Performance			Key Export Performance		
Group Size	Median (ms)	CV (%)	Group Size	Median (ms)	CV (%)
10	0.0034	13.94	10	0.005683	5.16
20	0.0050	10.24	20	0.00558	1.75
50	0.0080	15.92	50	0.005655	3.02
100	0.0132	13.32	100	0.005663	6.5

OpenMLS					**MLSpp**				
Encryption and Decryption Performance					Encryption and Decryption Performance				
Size	Enc (ms)	CV	Dec (ms)	CV	Size	Enc (ms)	CV	Dec (ms)	CV
100	0.0004	8.33	0.0003	8.82	100	0.0012	3.42	0.0011	2.58
1024	0.0011	13.13	0.0010	8.02	1024	0.0019	3.1	0.0016	3.23
2048	0.0019	10.42	0.0018	10.63	2048	0.0026	2.16	0.0021	4.52

OpenMLS				**MLSpp**			
Total Time Performance				Total Time Performance			
Group	100B	1024B	2048B	Group	100B	1024B	2048B
10	0.0041	0.0060	0.0073	10	0.0083	0.0094	0.0104
20	0.0057	0.0070	0.0086	20	0.0077	0.0093	0.0106
50	0.0089	0.0108	0.0121	50	0.0079	0.0093	0.0103
100	0.0144	0.0155	0.0170	100	0.0077	0.0089	0.0103

Appendix B. Threshold Signature Results

Table A4. Kryptology FROST_DKG_Ed25519 Results.

33% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	5.48	20.26	1.14	14.03	155.17	23.53	7.87	27.98
10	4	38.46	13.87	3.65	10.96	92.46	6.01	54.34	19.24
20	7	200.56	6.63	11.18	18.16	90.02	3.32	283.42	10.45
40	14	1394.17	1.85	52.28	11.55	101.96	5.79	1947.57	4.19
60	20	4873.94	3.25	142.62	12.21	86.95	4.08	5852.95	1.99
100	33	22,342.32	1.97	562.93	6.76	90.11	0.92	26,535.31	2.53
66% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	4	9.82	18.14	3.61	26.87	90.41	8.59	11.56	9.17
10	7	48.53	11.44	12.12	22.44	91.88	3.34	68.72	11.12
20	14	349.52	7.09	51.80	21.66	117.99	24.98	392.35	3.49
40	27	2468.97	1.79	248.81	5.95	89.21	3.17	276.89	9.91
60	40	9265.31	1.88	824.11	6.19	93.89	12.66	12,114.81	1.39
100	66	42,915.85	3.23	3471.01	3.61	150.893	4.18	54,228.44	2.42

Table A5. Kryptology FROST_DKG_secp256k1 Results.

33% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	35.41	20.63	7.55	19.66	938.47	33.19	50.63	17.95
10	4	213.88	8.52	23.36	17.60	529.09	23.56	263.06	12.13
20	7	1321.66	5.28	68.56	12.37	705.89	44.67	1436.12	8.20
40	14	10,077.84	2.53	284.52	10.91	549.06	40.49	11,697.41	4.21
60	20	37,907.52	14.50	1002.84	3.60	941.86	23.83	41,344.78	4.37
100	33	172,592.43	2.42	3996.30	1.80	993.40	30.52	173,948.96	1.55
66% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	4	50.29	12.91	22.09	20.23	878.70	43.42	79.17	9.39
10	7	313.18	8.35	72.29	14.11	646.48	33.1	427.51	8.20
20	14	2432.38	6.49	302.04	16.24	579.56	50.67	2724.29	4.13
40	27	18,516.92	2.85	1270.55	3.82	579.55	21.83	19,326.77	4.51
60	40	78,514.81	1.83	5144.81	5.11	941.26	25.08	80,340.13	5.45
100	66	315,975.24	1.68	15,975.60	1.71	905.33	26.08	347,080.33	1.50

Table A6. Zcash Foundation_FROST_DKG_Ed25519 Results.

33% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	6.29	1.85	0.61	9.05	155.17	8.43	6.65	4.54
10	4	38.46	4.49	1.14	5.87	92.46	31.01	39.09	5.20
20	7	252.03	1.46	2.26	8.53	90.02	11.87	236.09	3.38
40	14	1713.28	1.62	6.25	9.27	101.96	14.74	1708.33	1.91
60	20	5330.4	1.11	11.65	7.69	116.16	10.76	5446.51	0.63
100	34	24,357.32	0.5	28.91	6.64	115.21	50.9	24,865.36	0.67
66% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	4	9.5	1.38	1.12	1.81	90.41	11.28	10.51	1.5
10	7	58.3	2.13	2.21	7.46	91.88	5.75	60.46	3.8
20	14	423.24	0.88	6.35	1.93	117.99	20.37	433.71	2.61
40	27	3135.62	1.22	18.35	3.14	89.21	14.13	3205.45	0.82
60	40	11,089.14	1.11	37.98	5.33	114.27	17.97	10,477.19	0.78
100	67	46,831.99	1.38	101.66	3.5	110.42	379.14	47,916.47	0.45

Table A7. BankItalia_FROST_DKG_secp256k1 Results.

33% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	3.99	84.84	0.58	3.05	69.6	6.53	4.37	1.66
10	4	23.5	3.98	1.69	81.26	76.1	7.47	23.8	1.47
20	7	132	3.88	4.81	3.06	73.8	6.89	134	2.40
40	14	921	2.96	23.37	2.89	67.4	4.22	922	1.85
60	20	2802	26.27	57.63	4.76	69.5	5.2	2834	20.75
100	34	12,978	10.61	291.07	88.88	63.9	7.16	12,815	9.09

Table A7. *Cont.*

66% Threshold		DKG		Signing		Verification		End to End	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	4	5.92	3.39	1.70	81.64	67.5	4.54	7.38	79.79
10	7	33	2.77	4.67	1.23	65.1	5.76	36.2	3.29
20	14	230	5.00	21.74	3.55	67.6	53.38	239	4.34
40	27	1659	43.68	123.40	5.5	65.9	55.31	1761	40.27
60	40	5363	15.71	357.37	81.21	67.6	4.21	5472	14.14
100	67	27,044	4.13	1539.70	42.91	62.1	48.93	27,871	4.26

Table A8. BLSTRS_BLS_DKG Results.

33% Threshold		DKG		Signing		Verification		End to End	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	6.16	5.89	0.62	4.42	1548.47	2.97	8.47	3.87
10	4	45.46	1.43	1.05	7.07	1445.43	1.55	48.34	2.20
20	7	306.98	2.91	1.56	7.74	1498.12	7.18	303.97	1.86
40	14	2356.71	1.43	2.87	12.18	1517.14	2.93	2370.42	1.91
60	20	7515.98	1.07	4.12	22.32	1515.86	3.10	7646.57	0.67
100	34	35,762.23	2.92	6.82	7.50	1551.24	2.28	35,705.35	0.68
66% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	4	12.33	3.04	1.05	5.29	1573.08	8.51	14.99	3.62
10	7	78.99	1.96	1.59	10.85	1615.63	4.82	82.58	6.47
20	14	609.09	1.80	2.88	11.66	1516.66	3.00	604.81	1.11
40	27	4546.38	0.89	5.36	20.05	1551.48	8.69	4522.32	0.89
60	40	15,128.73	1.38	8.48	8.05	1507.28	1.62	15,297.59	0.75
100	67	70,244.19	1.57	14.73	15.05	1513.93	2.39	70,334.35	0.57

Table A9. BLSTTC_BLS_DKG Results.

33% Threshold		DKG		Signing		Verification		End to End	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	11.22	4.18	1.60	5.54	1438.87	10.73	1451.69	8.3
10	4	174.98	3.50	2.80	5.33	1340.23	5.81	1518.01	8.03
20	7	2083.02	2.33	4.41	14.7	1363.79	4.42	3451.22	4.52
40	14	34,777.16	2.94	9.28	11.98	1367.71	5.53	36,154.14	4.14
60	20	155,833.00	3.73	13.07	12.23	1375.75	6.79	157,221.82	5.1
100	34	1,244,927.00	3.72	6.36	13.03	1429.24	10.60	1,246,362.6	3.65
66% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	4	45.04	1.42	1.74	6.89	1402.74	14.33	1.40	5.55
10	7	530.69	1.09	2.51	5.13	1424.01	7.58	80.73	6.93
20	14	8422.45	0.90	5.85	6.01	1420.88	13.75	602.44	3.28
40	27	123,760.70	1.87	10.04	4.62	1375.18	6.65	4522.50	1.22
60	40	607,845.40	0.15	10.43	17.58	1411.36	5.46	14,990.82	0.89
100	67	–	–	–	–	–	–	–	–

– denotes results not available due to computational or runtime limitations.

Table A10. FROST_BankItalia Netsim_secp256k1 Results.

50 ms ± 10 ms + 2% Packet Loss (40% Threshold)									
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	160.4	22	129.7	–	0.2	–	350.4	13
10	4	343.3	15	216.9	–	0.2	–	500.5	12
20	8	457.4	5	272.4	–	0.1	–	722.1	6
40	16	1318	29	367.3	–	0.1	–	1685.7	26
120 ms ± 30 ms + 5% Packet Loss (40% Threshold)									
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	–	–	–	–	–	–	–	–
10	4	542.5	34	415.2	–	0.2	–	995.5	27
20	8	1125	23	493	–	0.2	–	1625.9	17
40	16	1960.5	2	1008.4	–	0.1	–	2967.7	14

– denotes results not available or not measured in the experiment.

Table A11. BLSTRS_BLS_Netsim Results.

50 ms ± 10 ms + 2% Packet Loss (40% Threshold)									
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	231.28	29.6	75.35	40.7	3.38	28.1	310.23	16.7
10	4	322.57	113.2	68.15	41.6	3.05	45.5	398.73	16.6
20	8	931.26	21.8	79.59	37.6	2.12	64.6	1050	46.5
40	16	5150	5	156.92	28.3	3.14	32.4	5290	4.3
120 ms ± 30 ms + 5% Packet Loss (40% Threshold)									
Nodes	t	Median	CV	Median	CV	Median	CV	Median	CV
5	2	381.53	11.5	157.11	44.7	138.5	47.9	534.21	16.9
10	4	552.88	14.7	177.25	40.1	125	36.1	755.79	13.2
20	8	1200	9.7	177.36	16.7	108.5	27.1	1400	8.5
40	16	5580	8.3	225.01	13.7	137	28.1	5810	7.4

Table A12. FROST Bulk Signing Performance (Zcash Ed25519).

Nodes	**Cycles**	**DKG**	**DKG CV**	**Signing**	**Signing CV**	**Verify**	**Verify CV**	**Sign+Verify**	**Total CV**	**Total Time**
20	10	811.52	14.5	1148.41	11.2	2.13	16.9	1203.06	4.46	2014.58
20	100	784.44	11.2	11,576.5	2.3	21.98	7.9	11,627.94	1.97	12,439.46
20	200	784.44	12.0	23,033.8	2.0	44.59	5.8	23,234.25	1.38	24,045.77
20	400	815.31	12.5	45,620	2.0	89.29	3.5	46,391.39	0.91	47,202.91
40	10	3055.57	9.0	1267.25	8.8	2.15	13.9	1392.77	4.48	4448.34
40	100	2986.55	11.4	12,693.5	3.1	22.03	7.2	14,771.06	1.87	17,826.63
40	200	2896.95	8.4	25,059.3	2.4	44.55	5.4	27,367.17	1.64	30,422.74
40	400	2969.82	8.7	50,335.9	2.3	89.26	3.5	52,488.77	0.80	55,544.34

Table A13. FROST BankItalia Bulk Signing Netsim (secp256k1).

Nodes	**Cycles**	**DKG**	**DKG CV**	**Signing**	**Signing CV**	**Verify**	**Verify CV**	**Sign+Verify**	**SV/DKG**	**Total Time**
20	10	452.6	13.9	2776.2	5.4	1.5	17.2	2777.7	1.72	3230.3
20	100	478.5	1.8	26,137.8	2.2	13.6	10.4	26,151.4	0.104	26,629.9
20	200	460.9	0.9	52,161.6	2.2	31.4	5.9	52,193	0.0295	52,653.9
20	400	546.5	0.5	103,719	1.3	60.9	5.9	103,780.1	0.009	104,326.6

Table A13. *Cont.*

Nodes	Cycles	DKG	DKG CV	Signing	Signing CV	Verify	Verify CV	Sign+Verify	SV/DKG	Total Time
40	10	937.5	19.4	3900.7	7.9	1.6	8.5	3902.3	0.85	4839.8
40	100	904	2.5	35,745.3	1.9	14.9	3.4	35,760.2	0.034	36,664.2
40	200	926.8	1.3	71,899.8	1.1	29.9	2.3	71,929.7	0.0115	72,856.5
40	400	814.5	0.5	147,239	0.8	55.3	2.7	147,294.4	0.00675	148,108.9

Table A14. BLSTRS Bulk Signing Netsim Results.

Nodes	Cycles	DKG	DKG CV	Signing	Signing CV	Verify	Verify CV	Sign+Verify	SV/DKG	Total Time
20	10	749.5	9.3	857.6	8.6	595.3	9.5	1452.9	1.94	2202.40
20	100	842.7	17.1	8589.1	4.9	5960.5	2.5	14,549.6	17.27	15,392.30
20	200	794.6	4.4	17,762.2	3.2	11,559.4	1.6	29,321.6	36.90	30,116.20
20	400	775.8	2.6	36,397.4	2.2	23,274.3	1.5	59,671.7	76.92	60,447.50
40	10	3631.1	5.7	1093.6	14.4	567.8	5.8	1661.4	0.46	5292.50
40	100	3451.4	4.3	11,329.6	2.8	5784	4.3	17,113.6	4.96	20,565.00
40	200	3505.6	8.9	23,041.7	3.1	11,519.5	2.2	34,561.2	9.86	38,066.80
40	400	3484.5	3.7	45,584.9	2.2	23,630.8	1.7	69,215.7	19.85	72,700.20

Table A15. Zcash DKG vs. Trusted Dealer Performance.

Nodes	DKG Time (ms)	Trusted Dealer (ms)	Ratio
5	285.56	73.85	3.87
10	462.77	84.46	5.48
20	847.58	101.05	8.39
40	3260.10	127	25.67
60	9116.86	147.13	61.96
100	31,335.87	162.69	192.61

Table A16. FROST BankItalia DKG vs. Trusted Dealer (secp256k1).

Nodes	DKG (ms)	TD Total (ms)	Ratio
5	154.93	64.42	2.40
10	267.15	68.91	3.88
20	415.72	75.84	5.48
40	901.48	149.78	6.02
60	1738.23	157.21	11.06
100	5184.08	167.85	30.89

Table A17. BLSTRS BLS DKG vs. Trusted Dealer.

Nodes	DKG Time (ms)	TD Time (ms)	Ratio
20	749.5	74.4	10.65
40	3631.1	156.4	27.16

References

1. Jangsher, S.; Al-Dweik, A.; Iraqi, Y.; Pandey, A.; Giacalone, J.P. Group Secret Key Generation Using Physical Layer Security for UAV Swarm Communications. *IEEE Trans. Aerosp. Electron. Syst.* **2023**, *59*, 8550–8564. [CrossRef]
2. Kwan, C.; Kish, L.; Saez, Y.; Cao, X. Low Cost and Unconditionally Secure Communications for Complex UAS Networks. In Proceedings of the IECON 2018—44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA, 21–23 October 2018; pp. 5895–5900. [CrossRef]

3. El-Zawawy, M.A.; Brighente, A.; Conti, M. Authenticating Drone-Assisted Internet of Vehicles Using Elliptic Curve Cryptography and Blockchain. *IEEE Trans. Netw. Serv. Manag.* **2023**, *20*, 1775–1789. [CrossRef]
4. Gupta, S.G.; Ghonge, M.M.; Jawandhiya, P.M. Review of Unmanned Aircraft System (UAS). *Int. J. Adv. Res. Comput. Eng. Technol. (IJARCET)* **2013**, *2*, 1645–1658. [CrossRef]
5. Shachtman, N. *Insurgents Intercepting Predator Video? No Problem*; Wired: San Francisco, CA, USA, 2009.
6. Swinney, C.J.; Woods, J.C. A Review of Security Incidents and Defence Techniques Relating to the Malicious Use of Small Unmanned Aerial Systems. *IEEE Aerosp. Electron. Syst. Mag.* **2022**, *37*, 14–28. [CrossRef]
7. Abdalla, A.S.; Marojevic, V. Security Threats and Cellular Network Procedures for Unmanned Aircraft Systems: Challenges and Opportunities. *IEEE Commun. Stand. Mag.* **2022**, *6*, 104–111. [CrossRef]
8. Limaye, Y. The Terrifying New Weapon Changing the War in Ukraine . *BBC News*, 28 May 2025. Available online: https://www.bbc.co.uk/news/articles/ckgn47e5qyno (accessed on 28 August 2025).
9. Civil Aviation Authority. *CAP 722: Unmanned Aircraft System Operations in UK Airspace—Guidance*; Technical Report 722; Civil Aviation Authority: Gatwick, UK, 2024.
10. Military Aviation Authority. *RA 1600: Remotely Piloted Air Systems (RPAS)*; Technical Report 1600; UK Ministry of Defence: London, UK, 2023.
11. Civil Aviation Authority. *CAP 3098: Guidance on Cyber Safety Objectives for Specific Category Operations*; Technical Report 3098; Civil Aviation Authority: Gatwick, UK, 2024.
12. United Kingdom. *Input to UN Secretary-General's Report on Lethal Autonomous Weapons Systems (LAWS)*; Technical report; United Nations Office for Disarmament Affairs: New York, NY, USA, 2024.
13. IETF MLS Working Group. The Messaging Layer Security (MLS) Protocol. RFC 9420. 2023. Available online: https://datatracker.ietf.org/doc/rfc9420/ (accessed on 11 March 2026).
14. Komlo, C.; Goldberg, I. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *Selected Areas in Cryptography*; Springer: Berlin/Heidelberg, Germany, 2020.
15. Tan, H.; Zheng, W.; Vijayakumar, P. Secure and Efficient Authenticated Key Management Scheme for UAV-Assisted Infrastructure-Less IoVs. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 6389–6400. [CrossRef]
16. Wang, Y.; Yang, D.; Zhao, Y.; Zhan, T.; Yang, Y.; Huang, W. Secure and Efficient Authenticated Key Management Scheme for FANET. In Proceedings of the 2025 4th International Conference on Cyber Security, Artificial Intelligence and the Digital Economy, Kuala Lumpur, Malaysia, 7–9 March 2025; pp. 125–131. [CrossRef]
17. Bhargavan, K.; Barnes, R.; Rescorla, E. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups: A Protocol Proposal for Messaging Layer Security (MLS). Research Report, Inria Paris. 2018. Available online: https://inria.hal.science/hal-02425247 (accessed on 11 March 2026).
18. Wallez, T.; Protzenko, J.; Beurdouche, B.; Bhargavan, K. TreeSync: Authenticated Group Management for Messaging Layer Security. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023.
19. Leon, A.; Britt, C.J. UXS Authentication and Key Exchange Requirements for Multidomain Operation and Joint Interoperability. Ph.D. Thesis, Naval Postgraduate School, Monterey, CA, USA, 2022.
20. Marstrander, E. Use of Messaging Layer Security in a Military UAV Swarm. Master's Thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2023.
21. Castro, M.; Liskov, B. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* **2002**, *20*, 398–461. [CrossRef]
22. Ippolito, C.A.; Krishnakumar, K.S.; Stepanyan, V.; Bencomo, A.; Chakrabarty, A.; Hening, S. An Autonomy Architecture Concept for High-Density Operations of Small UAS in Urban Environments. In Proceedings of the AIAA Scitech 2019 Forum, San Diego, CA, USA, 7–11 January 2019. [CrossRef]
23. Boneh, D.; Lynn, B.; Shacham, H. Short Signatures from the Weil Pairing. In *Proceedings of the Advances in Cryptology—ASIACRYPT 2001*; Boyd, C., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2248, pp. 514–532. [CrossRef]
24. Boldyreva, A. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Proceedings of the Public Key Cryptography—PKC 2003*; Desmedt, Y.G., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2567, pp. 31–46. [CrossRef]
25. Khaled, K.B.; Borsos, D.; Katona, J. Benchmarking Cryptographic Protocols for (IoT) Malware Defence. In Proceedings of IEEE 23rd World Symposium on Applied Machine Intelligence and Informatics (SAMI), Stará Lesná, Slovakia, 19 February 2025. [CrossRef]
26. Geisler, M. Digital Signatures in Wireless Sensor Networks. Master's Thesis, Aalborg University, Copenhagen, Denmark, 2008.
27. Leppänen, P. Performance Measurement of Cryptographic Protocols. Master's Thesis, University of Oulu, Oulu, Finland, 2014.
28. MLS Working Group. MLS Implementations. GitHub. 2024. Available online: https://github.com/mlswg/mls-implementations (accessed on 24 August 2025).

29. Rochford, P. Project GitHub Repository. 2025. Available online: https://github.com/RochP-Lab/Dissertation (accessed on 20 August 2025).
30. Kiefer, F. Post-Quantum OpenMLS 2024. Available online: https://blog.openmls.tech/posts/2024-04-11-pq-openmls/ (accessed on 15 July 2025).