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

Paul Rochford , William J Buchanan * , Richard Macfarlane , Madjid Golparvaran Tehrani

Posted Date: 9 December 2025

doi: 10.20944/preprints202512.0827.v1

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

 doi:10.20944/preprints202512.0827.v1

Article

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

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

Blockpass ID Lab, Edinburgh Napier University, Edinburgh
* Correspondence: b.buchanan@apier.ac.uk

Abstract

The decentralisation of autonomous Unmanned Aircraft Systems (UAS) introduces significant challenges for establishing secure communication and consensus in contested, resource-constrained environments. This dissertation 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 r esilience. 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, 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 (UAS) are aircraft that operate without a pilot or crew on board. They are frequently referred to as Remotely Piloted Aircraft Systems (RPAS), Unmanned Aircraft Vehicles (UAV) 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 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, UAS have become accessible to consumers and non-military users. UAS now range from small devices that can be held in the palm of your hand, with a range measured in metres, to traditional aircraft-sized platforms with a range measured in 1000 km. Regardless of the scale, the key components of a UAS are

 doi:10.20944/preprints202512.0827.v1

an aircraft 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 capability [8].

1.1. Regulatory Framework

In the UK, the Civil Aviation Authority (CAA) and Military Aviation Authority (MAA) are responsible for the regulatory oversight of civilian and military Unmanned Aircraft Systems, respectively [9,10]. Both agencies adopt the European Union Aviation Safety Agency's (EASA) three-tiered structure for categorising UAS (Open, Specific, Certified). These frameworks ensure that regulatory requirements are proportional to the level of risk associated with the type of UAS being considered. A key principle in the framework's approach is to remain technology agnostic, specifying what level of security/performance is required, without defining how to achieve it. This gives UAS developers the freedom to innovate and create new ways of securely deploying these technologies[11].

The maturation of AI and Machine Learning technologies means that autonomous UAS are now feasible. The emphasis of CAA and MAA regulations differs. The CAA focus is on ensuring deterministic behaviour by autonomous platforms in order to maintain an equivalent level of safety to crewed platforms[9]. In addition to flight safety, the MAA and wider Ministry of Defence emphasis is on compliance with international laws relating to armed conflict and ensuring a human remains 'in the loop' and responsible for the use of lethal weapons [10,12].

Recognising the potential risks resulting from autonomous UAS operations, the CAA encourages a *Secure by Design* methodology. This requires cybersecurity to be considered from the start of the design process and addresses both the network security of the system and the specific risk of AI and machine learning systems being manipulated in ways that could compromise their operational safety[11]. As such, there is a clear requirement for nodes within an autonomous UAS network to be able to communicate securely and for an architecture which can robustly handle compromised nodes attempting to maliciously undermine UAS operations.

One potential solution to these issues combines applied cryptography and distributed computing, focusing on methods to achieve Byzantine Fault Tolerance (BFT) and decentralised security. The Byzantine Generals' Problem [13] highlights the fundamental difficulty of achieving unanimous agreement among distributed entities when some participants may behave maliciously. Overcoming this requires cryptographic primitives that can ensure authenticity, integrity, and verifiability without relying on a central authority.

Fundamental to these solutions is the concept of secret sharing, which was independently defined in the works of Shamir [14] and Blakley [15]. Shamir's scheme, based on polynomial interpolation, offers a relatively simple method to divide a secret into multiple shares such that only a predefined threshold of shares can reconstruct the original secret, while fewer shares reveal no information. However, these initial secret sharing schemes required a "trusted dealer" to generate and distribute the shares, creating a single point of trust. This limitation led to the development of Verifiable Secret Sharing (VSS), pioneered by Feldman [16] and Pedersen [17]. Feldman's VSS introduced mechanisms for participants to verify the validity of their shares against public commitments, ensuring the honest behaviour of the dealer. Pedersen advanced VSS further and proposed DKG, where multiple parties collaboratively generate a shared secret key without any single party ever knowing the complete key [18]. This eliminates the trusted dealer entirely, achieving true decentralisation in key creation, but at the expense of increased communication overheads.

These advancements enable the construction of cryptographic schemes, such as threshold signatures, where a group can collectively perform a cryptographic operation (e.g., sign a message) only when a threshold of members contributes their secret shares. This collective signing capability provides

 doi:10.20944/preprints202512.0827.v1

a useful tool for decentralised authentication and verifiable group decision-making, offering a viable alternative to centralised models in the context of highly distributed and autonomous device networks, with clear relevance to contemporary problems faced by UK Defence as they increase the proportion of combat mass provided by unmanned vehicles [19].

This paper seeks to assess the performance of cryptographic libraries that could enable a decentralised key management and consensus mechanism for autonomous devices and reduce reliance on centralised control architectures. The main contributions of the paper are:

1. Provide a comprehensive comparison of current solutions for distributed key management and consensus.
2. Benchmark the performance of open source solutions, to demonstrate the selected technologies and analyse their performance. Identifying strengths, weaknesses, and areas for further improvement.
3. Assessing the viability of adopting existing solutions versus building a custom solution based on theoretical principles.

2. Related Work

This section will define the core cryptographic protocols that enable the project's key functions: secure, scalable group communication and decentralised consensus. All of these protocols are built on the fundamental concept of a cryptographic key. Much like a physical key secures a lock, a cryptographic key is a string of digital data (1's and 0's) that secures a message by applying an algorithm to it to make it readable only to those with the corresponding key. These keys can be public or private, and symmetric or asymmetric, but they all serve the same core function.

We will begin by exploring secret sharing, a technique for securely distributing a secret across multiple participants. Next, we will see how these secrets can be used to create digital signatures that verify a message's authenticity and integrity. Finally, we will define the methods used for secure group communication over otherwise insecure channels.

2.1. Secret Sharing

The three schemes outlined enable the division of a secret, for instance, a password, between multiple parties and then at a later time, a minimum threshold of those parties can reconvene and recreate the secret. Once implemented into usable libraries, these schemes require varying communication channels between the parties; the impact of those channels is not discussed here. Instead, the focus is on the mathematical principles that enable the schemes.

2.1.1. Shamir's Secret Sharing

Shamir's Secret Sharing is a threshold scheme where a secret is split into n shares such that any t shares can reconstruct the secret, but fewer than t reveal nothing. It has two phases: sharing and reconstruction. It relies on a trusted dealer to take the original secret and honestly follow the protocols to divide the secret into shares and likewise to reconstruct the secret.

2.1.2. Sharing Phase

Let $s \in \mathbb{Z}_q$ be the secret. The secret is shared as follows [14]:

1. Choose a random polynomial $f(x)$ of degree $t-1$ over $\mathbb{Z}_q$:

$$f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \pmod{q}$$

2. Generate and privately distribute n shares $y_j = f(j)$ to each participant $j \in \{1, \ldots, n\}$.

 doi:10.20944/preprints202512.0827.v1

Reconstruction Phase

Given any t shares from a set of participants J, the secret $s = f(0)$ can be reconstructed using Lagrange Interpolation:

$$s = \sum_{j \in J} y_j \prod_{\substack{k \in J \\ k \neq j}} \frac{k}{k-j} \pmod{q}$$

Shamir's scheme is *information-theoretically secure*, meaning that with fewer than t shares, the secret remains completely unknown, regardless of an adversary's computing power or time [14,20]. It is dependent on the honesty of the dealer and assumes that an adversary can eavesdrop on the various messages required to communicate the sharing, but not to modify them or the dealer's actions.

2.2. Verifiable Secret Sharing

Verifiable Secret Sharing (VSS) schemes enhance secret sharing by allowing participants to verify that a dealer has behaved honestly. This protects against a malicious dealer who might distribute invalid shares and ensures that a valid secret can be reconstructed when required.

2.2.1. Feldman's Verifiable Secret Sharing

Feldman's VSS is an extension of Shamir's scheme, which allows participants to verify that their shares originate from a single polynomial. The scheme's verifiability is based on homomorphic commitments and the computational difficulty of the discrete logarithm problem [16,20].

Let G be a cyclic group of large prime order q with a publicly known generator g. The secret s and the polynomial coefficients are elements of $\mathbb{Z}_q$.

Sharing Phase

The dealer performs the following steps to share the secret $s \in \mathbb{Z}_q$:

1. Choose a random polynomial $f(x)$ of degree $t-1$:

$$f(x) = s + a_1 x + \cdots + a_{t-1} x^{t-1} \pmod{q}$$

2. Compute and broadcast public commitments to the coefficients:

$$C_i = g^{a_i} \pmod{p} \quad \text{for } i = 0, \ldots, t-1 \quad (\text{where } a_0 = s)$$

3. Generate and privately distribute shares $s_j = f(j)$ to each participant $j \in \{1, \ldots, n\}$.

Verification Phase

Each participant j verifies their private share s_j against the public commitments by checking:

$$g^{s_j} \stackrel{?}{=} \prod_{i=0}^{t-1} (C_i)^{j^i} \pmod{p}$$

If the equality holds, the share is valid. This works because the right side expands to a commitment to $f(j)$:

$$\prod_{i=0}^{t-1} (C_i)^{j^i} = \prod_{i=0}^{t-1} (g^{a_i})^{j^i} = g^{\sum_{i=0}^{t-1} a_i j^i} = g^{f(j)} \pmod{p}$$

 doi:10.20944/preprints202512.0827.v1

Reconstruction Phase

Given any t valid shares from a set of participants J, the secret s is recovered using Lagrange Interpolation:

$$s = \sum_{j \in J} s_j \prod_{\substack{k \in J \\ k \neq j}} \frac{k}{k-j} \pmod{q}$$

Unlike Shamir's scheme, Feldman's VSS is not information-theoretically secure but is computationally secure. Its security relies on the assumption that the discrete logarithm problem is hard in the group G. The validity of that assumption in the face of Quantum Computing will be considered during the design section of this paper.

2.2.2. Pedersen's Verifiable Secret Sharing

Pedersen's VSS enhances Feldman's scheme to provide perfect hiding of the secret. While Feldman's scheme is computationally hiding, Pedersen's scheme is information-theoretically hiding. This is achieved by using a second random polynomial to blind the secret-carrying polynomial [18] [20].

The scheme operates in a group G of large prime order q. It requires two different generators, g and h, where the discrete logarithm of h with respect to g is unknown to all parties, including the dealer.

Sharing Phase

The dealer executes the following steps to share the secret $s \in \mathbb{Z}_q$:

1. Choose two random polynomials, $f(x)$ and $r(x)$, both of degree $t-1$:

$$f(x) = s + a_1 x + \cdots + a_{t-1} x^{t-1} \pmod{q}$$

$$r(x) = b_0 + b_1 x + \cdots + b_{t-1} x^{t-1} \pmod{q}$$

2. Compute and broadcast public commitments using both polynomials:

$$C_i = g^{a_i} h^{b_i} \pmod{p} \quad \text{for } i = 0, \ldots, t-1 \quad (\text{where } a_0 = s)$$

3. Generate and privately distribute a pair of shares $(s_j, t_j) = (f(j), r(j))$ to each participant $j \in \{1, \ldots, n\}$.

Verification Phase

Each participant j verifies their pair of shares (s_j, t_j) against the public commitments with the check:

$$g^{s_j} h^{t_j} \stackrel{?}{=} \prod_{i=0}^{t-1} (C_i)^{j^i} \pmod{p}$$

The verification works because the right-hand side is a commitment to the evaluation of both polynomials:

$$\prod_{i=0}^{t-1} (C_i)^{j^i} = \prod_{i=0}^{t-1} (g^{a_i} h^{b_i})^{j^i} = g^{\sum a_i j^i} h^{\sum b_i j^i} = g^{f(j)} h^{r(j)} \pmod{p}$$

Reconstruction Phase

Any group of t or more participants can reconstruct the secret. They pool their verified shares. The reconstruction only requires the primary shares, s_j. The blinding shares, t_j, are discarded after

 doi:10.20944/preprints202512.0827.v1

verification. Given t valid shares from a set of participants J, the secret s is recovered using Lagrange Interpolation on the s_j values:

$$s = \sum_{j \in J} s_j \prod_{\substack{k \in J \\ k \neq j}} \frac{k}{k-j} \pmod{q}$$

Pedersen's VSS is information-theoretically hiding because the commitment $C_0 = g^s h^{b_0}$ perfectly hides s thanks to the random b_0. The scheme remains computationally binding, as the dealer cannot find an alternative set of values to open the commitment without solving the discrete logarithm problem.

2.3. Distributed Key Generation

While Verifiable Secret Sharing (VSS) protects against a dishonest dealer, DKG removes the need for a dealer entirely. In a DKG protocol, a group of participants jointly generate a public key and a corresponding set of private key shares, without any single party ever knowing the full private key.

2.3.1. Pedersen's Distributed Key Generation

This protocol is a method for jointly creating a key pair by having all participants run Feldman's VSS in parallel. Each participant acts as a dealer for their own randomly chosen secret, and the final group secret is the sum of all secrets that were shared correctly [17,21].

The protocol operates in a group of prime order q with a public generator g.

Sharing Phase

1. Each participant P_i chooses a random polynomial $f_i(z)$ of degree t over $\mathbb{Z}_q$:

$$f_i(z) = a_{i0} + a_{i1}z + \cdots + a_{it}z^t \pmod{q}$$

   The value a_{i0} is participant P_i's individual secret contribution, which we can denote as x_i.
2. P_i broadcasts public commitments to the coefficients of their polynomial:

$$X_{ik} = g^{a_{ik}} \pmod{p} \quad \text{for } k = 0, \ldots, t$$

   The commitment to the secret contribution is $X_{i0} = g^{x_i}$.
3. P_i computes and privately sends the share $\bar{x}_{ij} = f_i(j) \pmod{q}$ to each other participant P_j.

Verification and Complaint Phase

1. Each participant P_j verifies the shares they received from every other participant P_i by checking if the share is consistent with the public commitments:

$$g^{\bar{x}_{ij}} \stackrel{?}{=} \prod_{k=0}^{t} (X_{ik})^{j^k} \pmod{p}$$

2. If the check fails for a share from P_i, participant P_j broadcasts a complaint against P_i.
3. If a participant P_i receives more than t complaints, they are disqualified. Otherwise, for each complaint from a participant P_j, P_i must broadcast the correct share $\bar{x}_{ij}$. If any revealed share is still invalid, P_i is disqualified.

Key Computation Phase

1. The set of all non-disqualified participants is established, often called the qualifying set.

 doi:10.20944/preprints202512.0827.v1

2. The final group public key, y, is computed by multiplying the initial commitments of all qualified participants:
$$y = \prod_{i \in \text{QUAL}} X_{i0} \pmod{p}$$
3. The final group secret key, x, is the sum of the individual secret contributions. This key is never computed in a single location.
$$x = \sum_{i \in \text{QUAL}} x_i \pmod{q}$$
Each participant P_j holds a private share of this final key, which is the sum of all the valid shares they received: $\sum_{i \in \text{QUAL}} \bar{x}_{ij}$.

2.4. Digital Signatures

Digital signatures are another fundamental cryptographic function that allows the authenticity and integrity of a message to be verified. Proving that a message has not been altered and has been sent by the claimed originator. This section will provide an introduction to signatures and a more detailed explanation of Schnorr, BLS and FROST signatures.

Digital signature schemes have three phases:

1. Key Generation. An algorithm that creates a public/private key pair (P,x). The private key is used for signing, and the public key is used for verification.
2. Signing. A process that uses the private key to produce a unique signature for a given message.
3. Verification. A process that takes a message, the signature and a public key as its input and will output "true" if the signature is valid for that message and public key (or "false" if verification fails).

The signing and verification phases are shown in Figure 1.

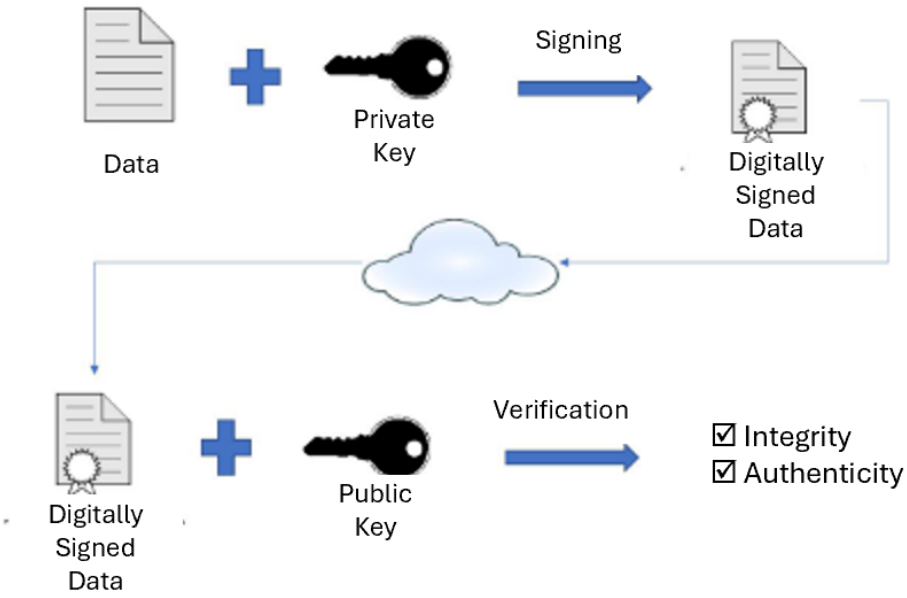

Figure 1. Signing and verification with digital signatures [22]

The security of the signature stems from the fact that it is impossible for an attacker to forge without knowing the private key. The signature schemes considered in this research all derive their security from the difficulty of the Discrete Logarithm Problem (DLP). Although this currently remains a difficult problem, the forecast imminent arrival of stable quantum computers threatens this difficulty

 doi:10.20944/preprints202512.0827.v1

[23]. The suitability of these signature schemes in the face of a quantum threat will be considered throughout this work.

2.4.1. Schnorr's Signature Scheme

Clauss Schnorr's signature scheme is a widely used protocol that was developed for use with resource-constrained devices and now forms the foundation for numerous signature protocols. It implements pre-processing, independent of the message being signed, to exploit idle processor time and reduce active signing time [24].

These signatures operate within a finite cyclic group where arithmetic is performed modulo a large prime number p. The group has a prime order q and is generated by a publicly known element g. Private keys and nonces are selected from the set $\mathbb{Z}_q$, and public keys are derived using exponentiation of g modulo p [25].

Key Generation

The signer selects a private key $x \in \mathbb{Z}_q$ and computes the corresponding public key $P = g^x \pmod{p}$.

Message Signing

To sign a message M, the signer performs the following steps:

1. Select a random nonce $k \in \mathbb{Z}_q$.
2. Compute the commitment $R = g^k \pmod{p}$.
3. Compute the challenge value $e = H(R, P, M)$, where H is a cryptographic hash function.
4. Compute $s = (k + e \cdot x) \pmod{q}$.

The signature is the pair (R, s).

Verification

To verify a signature (R, s) on a message M using the public key P, the verifier:

1. Computes $e = H(R, P, M)$.
2. Accepts the signature if and only if:

$$g^s \equiv R \cdot P^e \pmod{p} \tag{1}$$

2.4.2. BLS Signatures

The BLS signature scheme is a short, aggregate signature protocol based on bilinear pairings[26,27]. It is particularly useful in environments where signatures need to be short, such as low-bandwidth systems. The scheme's security relies on the difficulty of the Computational Diffie-Hellman (CDH) problem in certain groups, even when the Decision Diffie-Hellman (DDH) problem is easy.

The protocol operates within a pair of groups, G_1 and G_2, of prime order p, with a bilinear map $e : G_1 \times G_2 \to G_T$.

Key Generation

To create a key pair, a signer selects a private key, $x \in \mathbb{Z}_p^*$, at random. The corresponding public key is then computed as $v = g^x$, where g is a publicly known generator of the group. In the case of BLS signatures built on elliptic curves, this results in a public key v that is a point in the group G_1.

Message Signing

To sign a message, M, the signer performs the following steps:

1. Compute the hash of the message, $h = h(M)$, where h is a full-domain hash function that maps the message to a point in the group G_2.
2. The signature, σ, is then computed as $\sigma = h^x$, where x is the signer's private key.

 doi:10.20944/preprints202512.0827.v1

The resulting signature, σ, is a single element in G_2.

Verification

To verify a signature, σ, on a message M using the public key, v, the verifier:

1. Computes the hash of the message, $h = h(M)$, mapping it to a point in G_2.
2. Accepts the signature if and only if the following equation holds:

$$e(g, \sigma) = e(v, h) \tag{2}$$

where g is the generator of the group, e is the bilinear map, and v is the public key. This equation is a pairing-based check that verifies the signature without revealing the private key.

2.5. Secure Group Communications

One of the key challenges to enabling communication within a dynamic group is the secure distribution of a shared encryption key amongst its members. Traditional key exchange protocols such as Diffie-Hellman and Module Lattice-based Key Encapsulation Mechanism (ML-KEM) enable key exchange between a pair of clients. These pairwise connections can be utilised to implement a naive group solution, where a single client establishes a connection to every proposed member of a group and acts as a trusted dealer to share a group key.

However, this approach is time-consuming and resource-intensive, especially for large groups and those with a regularly changing membership. This approach is also dependent on the trustworthiness of the *dealer*. To address the limitations of this approach, a range of group key agreement protocols have been proposed. In this paper, we will show the progression from Group Diffie-Hellman Key Exchange to more scalable and secure methods such as Asynchronous Ratcheting Trees (ART) and Tree-based Key Encapsulation Mechanism (TreeKEM), which underpin modern protocols like Messaging Layer Security (MLS).

2.5.1. Diffie-Hellman Key Exchange

In 1976, Diffie and Hellman's key-exchange protocol showed how two parties (Alice and Bob) could derive a shared secret over insecure channels using collaborative exponentiation [28]. In their protocol, the parties publicly agree on a generator value g and a large prime number N.

They each choose their own private secret value, y for Alice and x for Bob. Alice calculates her public value a and Bob calculates his public value b:

$$a = g^y \pmod{N} \quad \text{and} \quad b = g^x \pmod{N}$$

They exchange the public values a and b with each other and can both then independently calculate the same shared secret key, K, as follows:

Alice's Calculation:

$$K = b^y \pmod{N} = (g^x)^y \pmod{N} = g^{xy} \pmod{N}$$

Bob's Calculation:

$$K = a^x \pmod{N} = (g^y)^x \pmod{N} = g^{yx} \pmod{N}$$

Since $g^{xy} = g^{yx}$, both parties arrive at the identical shared secret key K.

2.5.2. Group Diffie-Hellman

To extend this protocol to a group setting, Steiner et al developed Group Diffie-Hellman (GDH) key exchange, which allowed additional parties to pass their shared values sequentially in a ring

 doi:10.20944/preprints202512.0827.v1

around the group [29]. Each member adds their own secret exponent to the value passed from the previous member, effectively chaining the above calculations and allowing everyone on the ring to derive a shared key.

2.5.3. Tree Diffie-Hellman

While GDH allowed the sharing of a group key, it is a sequential protocol that scales linearly (O(n)), requiring significant computational and communication resources as a group expands. Tree-based Group Diffie-Hellman protocols were developed to improve efficiency. In this protocol, group members are organised in a binary tree structure, rather than in a ring. Here, the tree is structured into leaf, intermediate and root nodes as shown in Figure 2 (where nodes are identified by two digits that indicate their vertical and horizontal position in the tree).

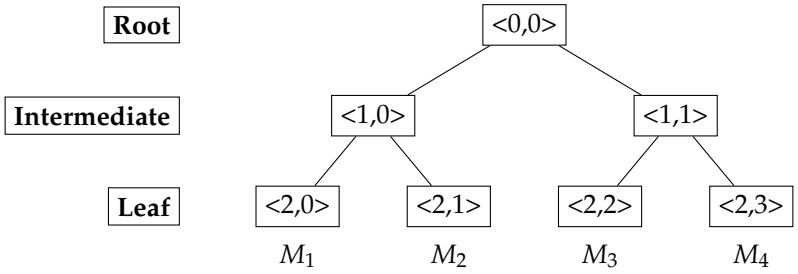

Figure 2. Tree-Based Group Diffie-Hellman key tree.

Each member is assigned a leaf node, every leaf and intermediate node above them has an associated secret key (K) and a public "blinded" key. These secret keys provide the entropy for the final group key, which each member derives independently using their secret key and the blinded keys. The blinded key (BK) is derived in the same way as the public share of a leaf's Diffie–Hellman exchange, i.e.,

$$BK = G^K \pmod{N}$$

To derive the group key, a node must know its own secret key and the blinded keys of its co-path nodes along the path to the root, as shown in Figure 3. Blind keys need to be broadcast to other members of the tree to allow them to derive the group key. When a new member is added (or departs) from the group, they are allocated as a new leaf node, updating the blind keys on their path to the root, requiring all members to recalculate the Group Key.

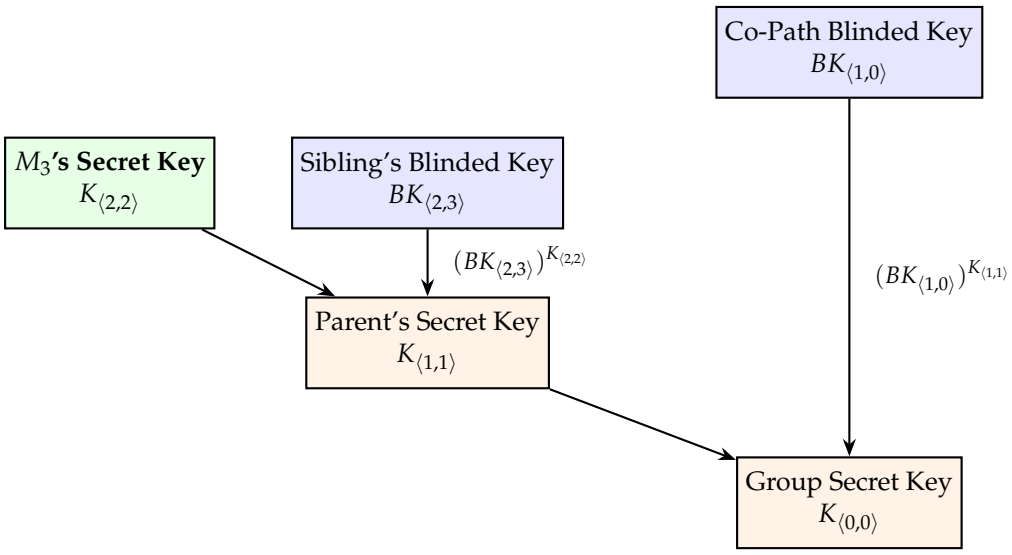

Figure 3. Example Key Derivation - M_3 derives the group key using ($K_{\langle 2,2\rangle}$) and the public blinded keys $\{\langle 2,3\rangle, \langle 1,0\rangle\}$.

 doi:10.20944/preprints202512.0827.v1

The use of a tree structure brings a significant increase in efficiency, from linear scaling with GroupDH to logarithmic scaling (O(log n)) with TGDH. However, the ability of the protocol to handle changes to a group membership and the impact of a compromised member on confidentiality remained sub-optimal. Requiring multiple rounds of communication for updates and without the Post Compromise Security provided by other protocols.

2.5.4. Asynchronous Ratcheting Trees

Cohn-Gordon et al. [30] highlighted the challenge of securely sharing a key for group messaging that can be scaled for large groups and can easily handle changes to group membership. Their Asynchronous Ratcheting Tree (ART) protocol was proposed as a fully asynchronous tree-based group key exchange protocol that allows clients to derive a group key without members needing to be online, that scales sub-linearly with the group size and provides robust security features, such as forward secrecy and post-compromise security.

ART defines three group operations: `Join`, `Leave`, and `Update`. The group is formed and expanded through a series of `Join` operations. No single member establishes the group key; rather, key derivation is collaborative. During an `Update`, a member changes their secret, allowing others to derive a fresh group key—enabling forward secrecy. Asynchronous operation is supported by an untrusted centralised server that hosts public keys and allows offline members to query updates upon reconnection. The system must maintain a strong ordering constraint to ensure consistent group key derivation.

The scheme operates in a cryptographic group G of large prime order q with a generator g, and uses a key derivation function H. The core cryptographic primitive is the Diffie-Hellman function: $DH(pk, sk) = pk^{sk}$.

Tree Structure and Key Derivation

Group key material is structured as a binary tree:

1. **Nodes and Members:** Each member u is assigned a leaf node. Each node v in the tree has a key pair (sk_v, pk_v), where $pk_v = g^{sk_v}$.
2. **Key Derivation Path:** For an intermediate node v, the secret key is derived via the DH exchange between its children L and R:

$$sk_v = H(DH(pk_L, sk_R)) = H(DH(pk_R, sk_L))$$

   This recursion continues up to the root node.
3. **Group Key:** The group key K is the secret key of the root node:

$$K = sk_{\text{root}}$$

Update Operation (Ratcheting)

Let member u be the updater:

1. **Leaf Update:** Member u generates a new ephemeral key pair: (sk'_u, pk'_u).
2. **Path Update:** Member u updates the secret keys along their direct path to the root. For each node v on this path, where v_{path} lies on u's path and v_{sibling} does not:

$$sk'_v = H(DH(pk_{v_{\text{sibling}}}, sk'_{v_{\text{path}}}))$$

3. **Broadcast and Key Recalculation:** Member u broadcasts the new public leaf key pk'_u and the updated public keys for sibling nodes along their path. Other members recompute the new root key:

$$K' = sk'_{\text{root}}$$

 doi:10.20944/preprints202512.0827.v1

2.6. Tree-Based Key Encapsulation Method

The ART protocol was a major step towards scalable, secure group messaging and underpins the Tree-based Key Encapsulation Method (TreeKEM). Developed by Bhargavan et al., this method uses the tree structure outlined above, but replaces Diffie-Hellman with an alternative key exchange method to reduce the computational complexity of the system and improve handling of concurrent update requests by group members [31].

TreeKEM uses a collision-resistant hash function, a public-key encryption mechanism, a pseudo-random key derivation function, and an authenticated encryption scheme. These primitives form the foundation of the protocol's functionality and through their elegant and sparing use, the creators achieved significant improvements over ART, particularly on recipient-side efficiency and more robust handling of concurrent updates.

The protocol defines four primary group operations: `CREATE`, `ADD`, `REMOVE`, and `UPDATE`. While the tree structure is nearly identical to that of ART, the method for deriving keys is fundamentally different.

Tree Structure and Key Derivation

TreeKEM uses a binary tree structure where members are assigned to leaf nodes. However, its key derivation method is non-contributive, which simplifies state updates.

1. **Nodes and Members:** Each member is a leaf on the tree. Each node in the tree has a secret key and a corresponding public key.
2. **Key Derivation Path:** Unlike ART, where an intermediate node's key is derived from a Diffie-Hellman exchange between both its children, a TreeKEM node's key is derived from a hash of the secret key of just one child, specifically the last child in that subtree to have performed an update.

$$sk_v = H(sk_{\text{child\_updater}})$$

   This non-contributive approach is what allows TreeKEM to more easily merge concurrent updates, as conflicting operations do not require complex resolution at the cryptographic level.
3. **Group Key:** The final group encryption key is derived from a chain of keys at the root of the tree, incorporating contributions from all updates over time to ensure security.

Update Operation

The update process in TreeKEM shifts the computational model from collaborative derivation (as in ART) to direct distribution via KEM. As an example, if member u is the updater:

1. **Leaf Update:** Member u generates a new key pair for their leaf node, (sk'_u, pk'_u).
2. **Path Update:** Member u calculates the new secret keys for all nodes on its direct path to the root by successively hashing its new leaf key: $H(sk'_u)$, $H(H(sk'_u)), \ldots$.
3. **Key Distribution via KEM:** Instead of broadcasting public values for others to re-calculate the path, member u encrypts the newly computed secret keys for the other members. For each node on its direct path, it encrypts the new secret key for the corresponding sibling node. This is done using the sibling node's public key.

$$\text{ciphertext} = \text{KEM.Encrypt}(pk_{\text{sibling}}, sk'_{\text{parent}})$$

   This message is then sent to the members of that sibling group.

This change significantly reduces the work for receiving members. To receive an update, a member only needs to perform one KEM decryption and a series of fast hash operations, versus the multiple, more computationally intensive Diffie-Hellman operations required in ART. This makes TreeKEM more efficient, particularly for recipients.

 doi:10.20944/preprints202512.0827.v1

2.7. Byzantine Fault Tolerance

Modern autonomous systems are made up of numerous components. To be resilient, they must cope not only with the complete failure of a component, but also with faulty or unreliable components that provide erroneous signals, or components that have been compromised by an adversary and are being directed to provide deliberately harmful signals[32].

A useful metaphor for this situation is the Byzantine Generals Problem. Here, we imagine several divisions of the Byzantine Army surrounding a city. Each division is commanded by a general, and the generals are connected to each other by messenger. The generals may be loyal or they may be traitors, with the traitors seeking to prevent the loyal generals from agreeing on a successful plan of attack.

Lamport et al. [13] proved that in such a situation, a simple majority vote would be insufficient to ensure that an honest majority could reach consensus in the presence of Byzantine faults. With *oral messages*—where a general can send conflicting messages to different recipients—a system with m traitors requires at least $3m + 1$ total participants to achieve consensus. When *signed messages* are used—ensuring authenticity and integrity—a system can tolerate up to $\left\lfloor \frac{n-1}{2} \right\rfloor$ traitors, provided that honest nodes can communicate either directly or indirectly with one another.

These theoretical limitations directly impact the ability of Verifiable Secret Sharing (VSS) and (DKG) schemes to tolerate Byzantine faults. Asynchronous systems, where paths between nodes are not guaranteed, can behave like systems with *oral messages* and are typically limited to fault tolerance of $m < \frac{n}{3}$, where m is the number of traitors and n is the total group size. For systems with public commitments or *signed messages*, that tolerance increases to $m < \frac{n}{2}$.

3. Related Work

The development of these cryptographic primitives into mature libraries and algorithms has taken many years and seen numerous weaknesses identified and mitigated. This section will chart some of these developments and provide a clear understanding of the current state of the art within the fields of group key distribution and threshold cryptosystems, with a focus on their applications within distributed systems, such as UAS.

3.1. Key Distribution in UAS

As research into Internet of Things (IoT), Internet of Vehicles (IoV), multi-vehicle UAS swarms and systems that combine all three has developed. The challenge of providing secure communications between devices to enable autonomy has been approached in multiple different ways, largely driven by the underpinning architecture and its constraints. These approaches have adopted both purpose-built designs as well as adapted pre-existing protocols; the efficacy of these approaches and lessons for the development of our implementation are outlined next.

Although the Russian invasion of Ukraine has led to both sides using commercial cellular technology to provide C2 links for their UAS, [33,34], Abdalla et al. [7] highlight the difficulty of securing cellular networks to enable civilian UAS operations. However, their work focuses on UAV to Ground Control Station (GC) connections, and suggests authentication and latency times that would be unsuitable for drone-to-drone communication. What is more, for UAS intended to operate in a degraded or denied EM environment, reliance on an adversary's cellular network would be a serious tactical weakness.

3.1.1. A Centralised Approach

Noguchi [35] devised a key-sharing system that utilised Shamir Secret Sharing to achieve an order of magnitude improvement in encoding times compared to RSA for resource-constrained devices (Raspberry Pi 3s). The approach relied on a trusted central architecture and pre-shared secrets to authenticate devices, with no authentication by the management nodes to verify user devices. This shows that the very lightweight approach to authentication and the use of Message Queuing Telemetry Transport MQTT, which is common for constrained devices, would not be suitable for our use case.

 doi:10.20944/preprints202512.0827.v1

Their methodology identified key performance metrics that were visible in multiple papers, namely encoding delay, reconstruction delay and total key-sharing delay.

Building on a more robust architectural model, Tan [36] approached the challenge of group key distribution in an innovative way, with a trusted dealer using certificateless authentication, disseminated during a configuration phase by a central Trust Authority pre-flight, to verify nodes. Similar to Shamir's sharing scheme, their proposal utilises the mathematical properties of polynomials, but this time employing the Chinese Remainder Theorem CRT to enable individual reconstruction of the group secret key. Their approach depends on a powerful tethered UAV as the trusted dealer, removing the power limitation inherent to most UAS swarms. While this is feasible for enabling devices in an Internet of Vehicles IoV architecture, it is absolutely unsuited to our use case.

3.1.2. Shifting to Decentralisation

Mitigating the reliance on a single management node, Wang et al [37] expanded the CRT polynomial technique with the addition of a token system. This reduces the computational overhead of the system and gives it greater utility across a wider range of UAV use cases; however, for larger swarms (100+), the token size would remain very large in order for the CRT to function correctly. A multi-stage, threshold approach to authentication included Shamir Secret Sharing to provide added resilience, but requires a threshold of management nodes to be operational during authentication. The batch injection of certificateless identities expedites configuration times and has merit, but constrains the ability for nodes to be added on an ad-hoc basis if a mission unexpectedly needed a capability that had not been included during the configuration phase.

Moving away from purely cryptographic solutions, Jangsher et al's [1] approach was to introduce dedicated hardware to their system (Physically Unclonable Functions (PUFs) and emulators (PUFe's). During a configuration phase, devices are provided with the digital fingerprint for a cohort of devices. Multiple overlapping cohorts will form a swarm. During key sharing, a pair of devices initiates a connection, using the PUF and PUFe to verify each other. The devices then create a pairwise key to secure their connection using a physical measurement of their link (i.e signal strength or similar). One device at a time, UAVs are added to the swarm, and gradually, the group's shared key is formed. The use of PUFs and physical measurements removes the requirement for Public Key Cryptography (PKC), reducing the cryptographic computation required by the devices. However, the time to form a swarm increases linearly with its size and can stall with mid-sized groups (those over 10).

In contrast to hardware-dependent models, Yang [38] proposes a Group Authenticated Key Exchange protocol based on Elliptic Curve Diffie-Hellman ECDH and authenticated using Boneh-Lynn-Shacham BLS short signatures. The protocol utilises authentication to mitigate Man In The Middle MITM attacks and the authors highlight its relevance to UAS Swarms, where devices may be captured by an adversary or temporarily lost. However, it uses a member of the group to act as the Group Controller (GC), which then coordinates the key generation and update process. The cryptologic burden of creating keys and then distributing them securely is placed on the GC, and this becomes a significant challenge for larger swarms that have a dynamic composition with significant churn.

3.1.3. A Modern Group Authenticated Key Exchange Protocol

Taking a broader perspective, Leon and Britt reviewed eight open and proprietary protocols to assess their suitability for use in military Unmanned Systems [39]. Due to its provision of FS, PCS, support for asynchronicity and logarithmic scaling efficiency, MLS was assessed to be the most suitable. MLS was successfully tested with a small three-member group of UAS and USV (Unmanned Surface Vessel), proving the utility of the protocol in this environment. The study identified a number of areas for future work that limited the resilience of their implementation, specifically with their manual approach to Authentication and Delivery Services.

During the development of MLS, Wallez et al. proved the ability of MLS to robustly maintain group authentication and integrity, making it highly suitable for use by UAS operating in a contested

 doi:10.20944/preprints202512.0827.v1

environment [40]. Specifically, the ability of the protocol to ensure all legitimate members have a consistent and authenticated view of the group's membership is resilient to an attacker attempting to insert an illegitimate member and is capable of securely handling a high rate of member turnover/churn.

Finally, offering the most comprehensive solution for dynamic groups, Marstrander's [41] research took a different approach, exploiting the IETF ratified Message Layer Security (MLS) Protocol and existing libraries and applications to develop the Flamingo MLS system for key management within meshed UAS networks. MLS utilises the TreeKEM scheme for securely generating encryption keys across large groups. While still requiring centralised nodes for a coordinating function, the cryptographic load is distributed across the nodes, minimising the computational load on a single device. It is commonly used within secure messaging apps to protect group chats, and is optimised for the rapid distribution of keys across large group sizes and can robustly handle both frequent addition and revocation of group members. Marstrander's work, grounded in the reality of military UAS operations, recognised the challenges of identifying a compromised node, the requirement for a decentralised consensus mechanism to respond coherently to compromised nodes and proved a decentralised DS using the Totem protocol. This novel study proved the utility of MLS on a small network of resource-limited devices, while identifying a number of limitations within their experiment that warrant further investigation.

3.2. Threshold Cryptosystems

Among many possible applications, threshold cryptosystems can form the foundation for secure consensus or voting, as they allow a threshold t out of n members to jointly complete a cryptographic operation — for example, applying a digital signature to a proposal. A common way to realise such systems is through polynomial-based Distributed Key Generation (DKG), which ensures that no single party ever learns the full private key. On top of this shared foundation, different signature constructions can be used to enforce the threshold property. This research focuses on two specific approaches: bilinear-pairing-based signatures, represented by BLS, and Schnorr-based protocols, represented by FROST.

NASA's Ames Research Centre's SAFE50 Autonomy Reference Architecture defines the extensive range of hazards, environmental effects, and operational considerations that necessitate a consensus approach [42]. Overall, the UAS must navigate an environment filled with static and dynamic objects in three dimensions, all while contending with environmental challenges and potential system failures. In our conceptual architecture, multiple devices will provide different sensor capabilities and perspectives. While SAFE50 does not propose specific cryptographic mechanisms, this work highlights threshold signatures as one potential means of increasing confidence in the decisions being made by autonomous UAS.

3.3. Distributed Key Generation

The starting point for these systems is DKG. Section 2.3 showed that Pedersen's development of DKG enables a group to jointly create a key pair without relying on a centralised trusted dealer. The core of the protocol is that each participant generates a random polynomial of degree $t-1$, where t is a predefined threshold (equal to n in the original theorem). By reducing t, any quorum of at least t participants can work together to perform cryptographic operations—without ever reconstructing the private key in full. This scheme can tolerate up to $m < n/2$ malicious parties, where m is the number of malicious parties. As with all DKG protocols, multiple rounds of communication are required to ensure honest behaviour. While this decentralised approach aligns with our design objectives by removing any single point of trust, it is more costly in terms of communication and setup time compared to the trusted dealer model. This trade-off will be highlighted during the benchmarking process, where both decentralised DKG and trusted dealer configurations are evaluated, with libraries supporting them.

 doi:10.20944/preprints202512.0827.v1

3.4. BLS Signing

While Pedersen's DKG relies on polynomial secret sharing for key generation, in 2003, Boldyreva extended the BLS signature scheme to apply a similar threshold principle using bilinear pairings. In this pairing-based approach, each participant P_i chooses a random secret value $x_i \in \mathbb{Z}_p$ and computes a public commitment $X_i = g^{x_i}$. The group public key V_{group} is the product of all individual public commitments. As with the polynomial-based scheme, the full private key is never reconstructed.

To sign a message M, each of the t participants computes a partial signature using their private share:

$$\sigma_i = H(M)^{x_i}$$

The final signature is aggregated by multiplying the partial signatures:

$$\sigma_{\text{final}} = \prod_{i=1}^{t} \sigma_i$$

To verify, a verifier checks:

$$e(g, \sigma_{\text{final}}) = e(V_{\text{group}}, H(M))$$

This BLS threshold scheme tolerates up to $m < n/2$ malicious parties, which is the theoretical optimum and functions with a single round signing phase, an improvement over the multiple rounds needed by other schemes. However, the pairing function required for verification and aggregation is computationally intensive[43].

3.5. FROST Signing

BLS offers a single-round signing phase (albeit with computationally expensive pairing operations for verification and aggregation), whereas earlier robust threshold Schnorr protocols, such as those described by Gennaro et al., required three signing rounds [44].

These signing rounds are a significant cost for resource-constrained networks. Komlo and Goldberg sought to design a threshold signature scheme that maintained security while improving performance [45]. Their Flexible Round-Optimised Schnorr Threshold (FROST) signatures improved the state of the art and showed it possible to have either a two-round signing protocol, or a single-round signing protocol with pre-processing and a centralised, semi-trusted signature-aggregator (due to better alignment with our design objectives, we will focus on the decentralised version).

The FROST protocol consists of three main stages: Key Generation, Signing and Verification. FROST typically employs Pedersen's DKG with an added Zero Knowledge Proof (ZKP) to protect against rogue-key attacks, to provide participants with a long-lived secret key share and produce a public key that is known to all participants.

In the first round of signing, each participant generates two single-use nonces and their corresponding public commitments. The commitments are broadcast to all participants. In the second round, the broadcast commitments are used to calculate a group commitment and a binding value that ties the message, participants and commitments together. They then use their secret share, nonces and the binding value to compute their signature share. The signature shares can then be broadcast and aggregated into a final group signature. Verification is identical to the single-party Schnorr signature verification method. In summary, BLS minimises interactive latency at the cost of expensive pairing-based verification, while FROST reduces computational load but requires an additional round of communication. Both schemes were therefore selected for benchmarking to evaluate how these trade-offs manifest in practice.

4. Methodology

This section applies concepts from the literature review 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

 doi:10.20944/preprints202512.0827.v1

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 4 .

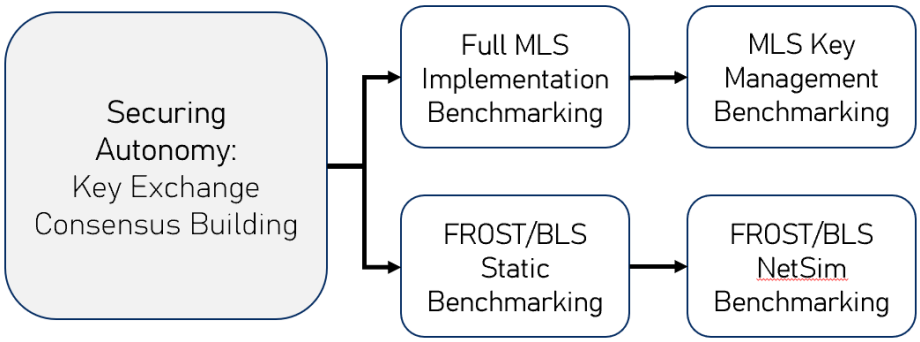

Figure 4. Research Scope

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 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 [46–48].

4.0.1. 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.

4.0.2. 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.

4.0.3. 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.

4.1. 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

 doi:10.20944/preprints202512.0827.v1

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.

4.1.1. 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.

4.1.2. MLS Architecture

A simple MLS architecture is shown in Figure 5. 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.

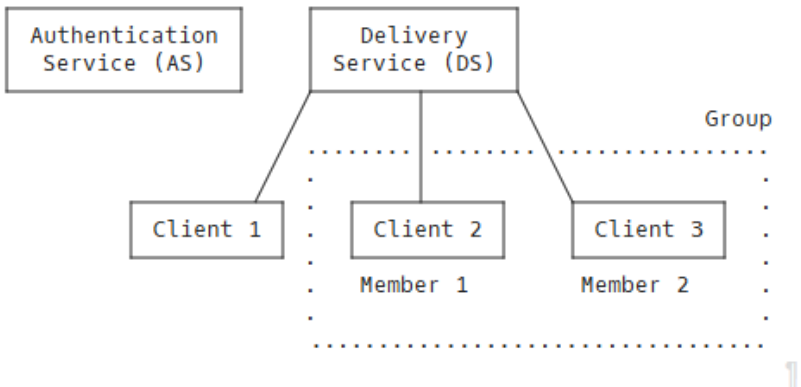

Figure 5. Example MLS Architecture [49]

The MLS RFC supports the use of PKC as an acceptable means of implementing a DS, and this approach is compatible with our pre-mission configuration phase concept. The DS is responsible for managing group membership, 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

 doi:10.20944/preprints202512.0827.v1

employed in a mobile messaging app, this would be a fixed server. This centralised approach could be replicated within our UAS architecture, with replication providing resilience to a DS becoming unavailable. However, the RFC supports the concept of a decentralised peer-to-peer DS, and previous work by Marstrander [41] 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.

4.1.3. MLS Operations

This extensive capability, defined in IETF RFC 9420, is provided by a complex arrangement of binary tree structures and multiple encryption keys. The foundational theoretical concepts are discussed in the preliminary chapters, and a full explanation can be found in the RFC [49]. However, to understand the research undertaken within this project, a deeper understanding of the following operations will be beneficial:

1. **Group Formation** This is a computation and communication-intensive process for large groups and should be completed during the pre-mission configuration phase. This operation establishes the binary tree structures, populating the leaf nodes with identity information and initial key material that produce the epoch and synchronous keys.
2. **Add Member** This operation tests the cost of integrating a new client into an existing group's tree structure. It generates an `Add` proposal for the new client, creates a `Commit` message for the proposal, then re-computes the groups tree to accommodate the new leaf node. Finally, it generates a `Welcome` message for the new client. This contains all the cryptographic material they would need to join the group and be part of the current epoch.
3. **Remove Member** This operation tests the costs of removing a client from a group and updating the remaining members. It generates a `Remove` proposal for the target client. Then creates a `Commit` message that includes this proposal and then recomputes the cryptographic tree to remove the target client's leaf node.
4. **Update Member** This operation tests the cost for a single client to initiate an update. This has the effect of shifting to the next epoch and rekeying the system. It generates a new private-public key pair for the client. Recomputes the member's cryptographic data and updates its leaf node in the tree structure. Constructs a `Commit` message to update the group.
5. **Process Update** This operation tests the costs for a client to receive and apply an update message. When the client receives a `Commit` message, it must process it to synchronise its group state. The client must authenticate the `Commit` message by verifying the signatures within the message, then recompute and update its local copy of the group's cryptographic tree to reflect the change. Derive the new group keys for the next epoch.
6. **Protect Message** This operation tests the cost for a client to encrypt a message for the group. This requires the client to generate an encryption key using the Secret Tree, encrypt the message and then sign the message.
7. **Unprotect Message** This operation tests the cost for a client to decrypt a message sent by another member of the group. This requires the client to generate an encryption key (as per Protect Message) and then use it to decrypt and authenticate the message.
8. **Key Export** This operation tests the costs for a client to generate a symmetric encryption key and present it via an API call for use by another application. It generates this key from the Secret Tree. This is computed 'locally' by the client, requires no inputs from other clients and is a relatively efficient operation.

4.1.4. Library Selection

The MLS Working Group maintain a list of active MLS libraries [50]. The Cisco MLSpp library has been proven to work on a UAS network in previous work [41], so a second library was selected for comparison. The aim is to identify a library less susceptible to the performance issues Marstrander

 doi:10.20944/preprints202512.0827.v1

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.

4.1.5. 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. Figure 6 shows the test variables.

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

Figure 6. MLS Benchmark Variables

4.1.6. 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.

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.

4.1.7. 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 [51].

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

 doi:10.20944/preprints202512.0827.v1

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.

4.2. 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. Figure 7 compares each protocol's computation and communication processes.

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 commitments 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 don't 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 their 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 the commitments from all	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 2 round DKG with a reduced level of security.

Figure 7. Threshold Signature Rounds

 doi:10.20944/preprints202512.0827.v1

4.2.1. Library Selection

Figure 8 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.

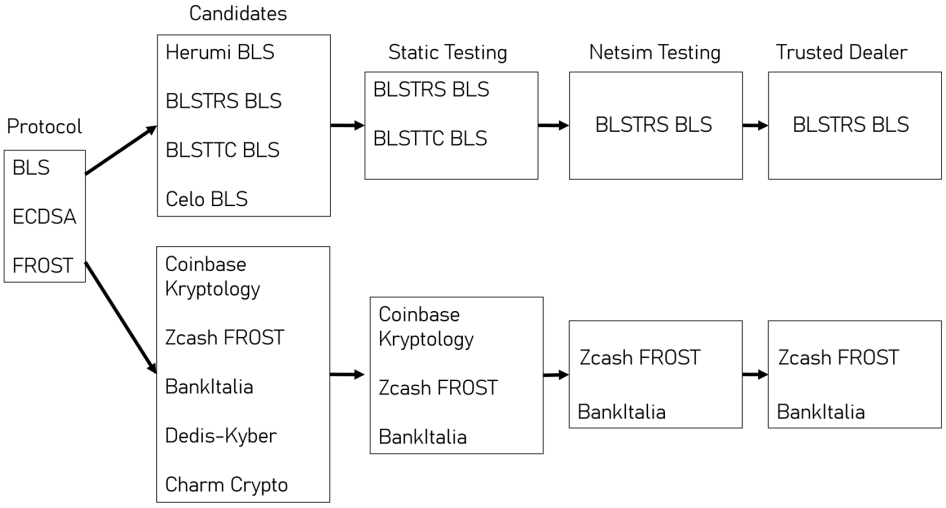

Figure 8. Threshold Signature Benchmark Development

Starting with static benchmarks that isolated the library's cryptographic functions and ran tests that would allow 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 the performance cost of the decentralised key generation to be quantified.

4.2.2. 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 UAS 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 1.

 doi:10.20944/preprints202512.0827.v1

Table 1. 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 -bls	Rust	BLS12-381	DKG	Google Benchmark	BTreeMap

4.2.3. 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.

4.3. 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 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 2.

 doi:10.20944/preprints202512.0827.v1

Table 2. Network Performance Tests

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

4.4. 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 the overhead of decentralisation to be quantified.

4.4.1. Benchmark Environment

Microsoft Visual Studio Code with integrated GitHub AI Copilot was used for code development. This allowed 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 [51].

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 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 24hrs) 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.

 doi:10.20944/preprints202512.0827.v1

5. 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 3 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 3. Completed Benchmark Summary

Protocol	Library	Description
MLS	OpenMLS	Key Distribution & Msg Encryption modes
MLS	MLSpp	Key Distribution & Msg Encryption modes
FROST	Kryptology	Ed25519 DKG static
FROST	Kryptology	Secp256k1, DKG static
FROST	Zcash	Ed25519 DKG static & bulk sign & TD
FROST	BankItalia	Secp256k1 DKG static & netsim & bulk sign & TD
BLS	BLSTTC	BLS12-381 DKG static
BLS	BLSTRS	BLS12-381 DKG static & netsim & bulk sign & TD

5.1. 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.

5.2. OpenMLS Results

5.2.1. Full MLS Mode Benchmark

OpenMLS showed consistently high levels of performance across all cipher suites. Increasing client number from 10 to 100 had a relatively small impact on performance, with the `Add Member` operations increasing by a factor of 1.5x to 2.4x 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.

OpenMLS

Ciphersuite	Group Size	Add Member (ms)	Update Member (ms)	Remove Member (ms)	Process Update (ms)	Msg (100B) (ms)	Msg (1024B) (ms)	Msg (2048B) (ms)
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

Figure 9. OpenMLS Full Results

5.2.2. 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 Figure 10. This shows performance reducing as both message size and group size increase. Key export times

 doi:10.20944/preprints202512.0827.v1

dominate these results, for 100 client groups a performance reduction between 2.3x and 3.5x was observed relative to the 10 client group. A smaller reduction in performance was seen as message size increased from 100B to 2048B, between 1.2x and 1.8x.

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

Figure 10. OpenMLS Key Management Mode Results

5.3. MLSpp Results

5.3.1. 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 3x as group sized increased from 10 to 100. `Update Member/Remove Member/Process Update` showed even larger increases, up to 8.3x 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 1.6x as message size increased from 100B to 2048B (and 2.2x in the worst case.). These results are shown in Figure 11

MLSpp

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	33.32	213.44	25.74	212.01	5.637	5.924	6.405
	100	37.41	431.62	32.04	400.90	4.855	5.040	5.703

Figure 11. MLSpp Full Mode Benchmark Results

5.3.2. 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 2.1x and 1.9x 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 1.65x advantage was observed for MLSpp. Figure 12 shows the end to end results for MLSpp.

 doi:10.20944/preprints202512.0827.v1

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

Figure 12. MLSpp Key Management Mode Benchmark Results

5.3.3. Open MLS vs MLSpp

OpenMLS comprehensively outperforms MLSpp for all aspects of the MLS Full Mode benchmark. Figure 13 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 [41], that MLSpp suffered data marshalling issues and was a sub-optimal choice for this type of application.

Figure 13. OpenMLS vs MLSpp Performance

5.3.4. Full vs Key Management Mode

The key export function allows 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 14, where the time taken for OpenMLS to complete one encryption cycle is 20x greater for a 1024B message and a group size of 10. Both libraries showed similar stand alone encryption/decryption performance (within one microsecond of each other), although MLSpp showed

 doi:10.20944/preprints202512.0827.v1

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.

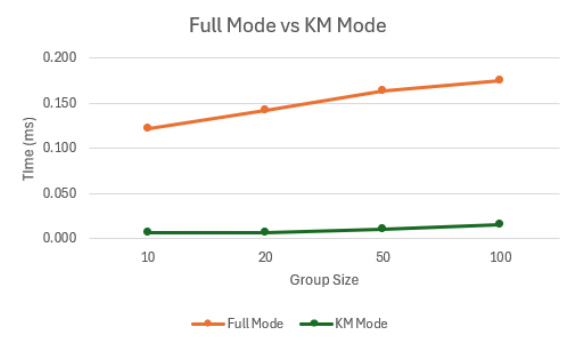

Figure 14. Full mode vs Key Management Mode

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.

5.3.5. 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 15, the top 3 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 [39,41] 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 [52]. 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

 doi:10.20944/preprints202512.0827.v1

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.

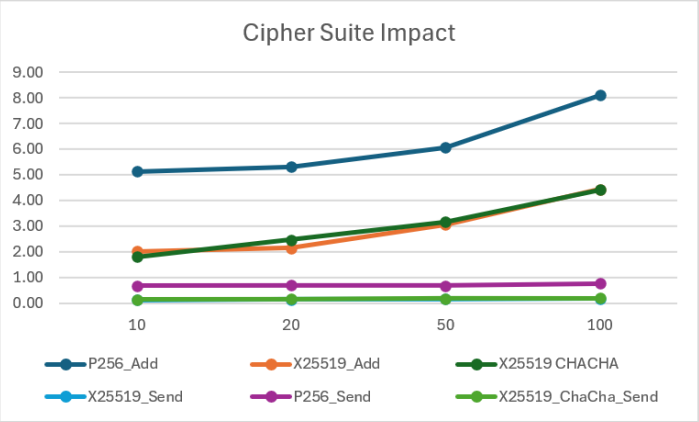

Figure 15. OpenMLS Cipher Suite Performance

5.3.6. 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 managmenet 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. However, its inclusion within the benchmark would have further informed the development of that concept and is a small missed opportunity.

5.4. 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 chapter where it is appropriate.

5.4.1. 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 developer' implementation. Figure 16 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 3000x reduction in performance (Bank of Italia) as the number of nodes increased from 5 to 100 and the worst suffering a 5000x 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

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 December 2025 doi:10.20944/preprints202512.0827.v1

curves and the quality of the overall implementation have a bigger impact on performance than using a fast or slow curve.

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	11697.41	1708.33	922.00	2370.42
	60	5852.95	41344.78	5446.51	2834.00	7646.57
	100	26535.31	173948.96	24865.36	12815.00	35705.35
Threshold	Nodes	FROST - DKG				BLS-DKG
		Kryptology-Ed25519	Kryptology - Secp256k1	Zcash-Ed25519	BankItalia - Secp256k1	BLSTRS - BN12
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	19326.77	3205.45	1761.00	4522.32
	60	12114.81	80340.13	10477.19	5472.00	15297.59
	100	54228.44	347080.33	47916.47	27871.00	70334.35

Figure 16. Results - Static End to End

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 17.

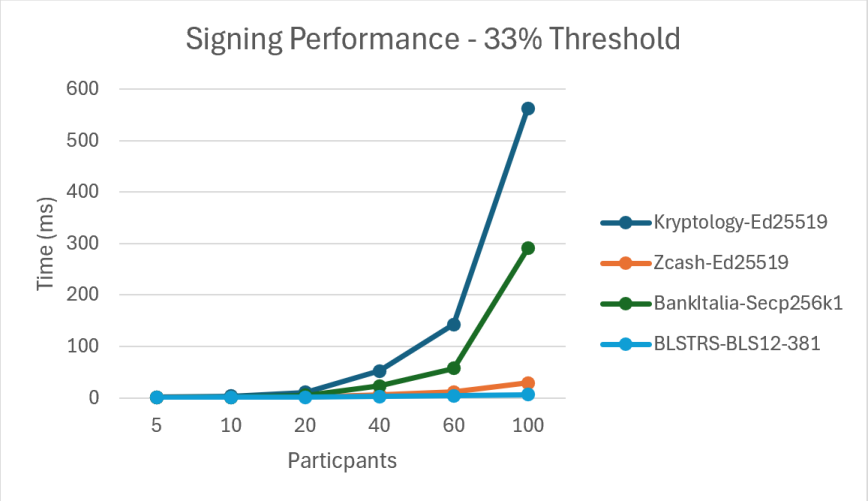

Figure 17. Results - Signing Performance

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

 doi:10.20944/preprints202512.0827.v1

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 18.

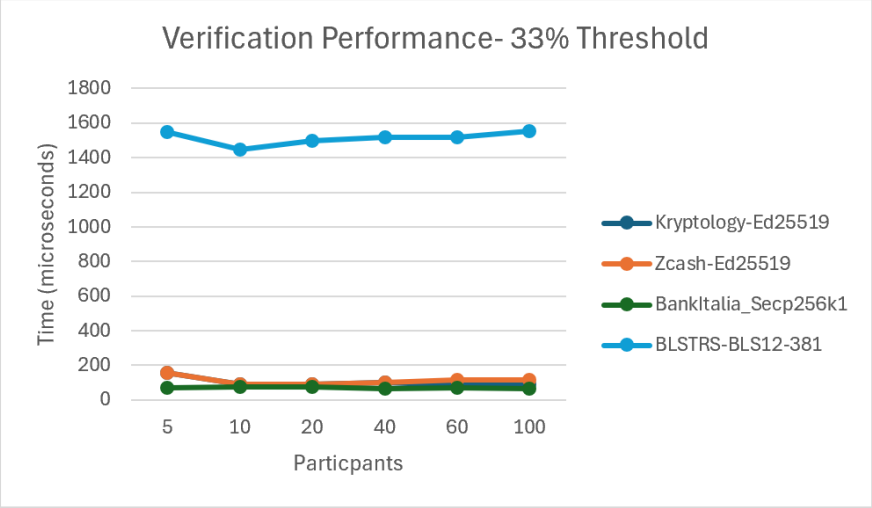

Figure 18. Results - Verification Performance

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 19 shows the static cost of 1 DKG cycle with 20000 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.

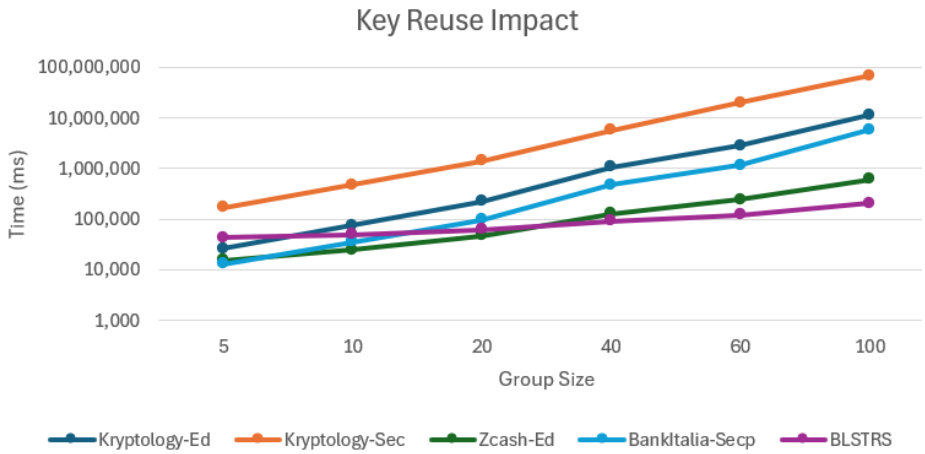

Figure 19. Results - Key Reuse Impact

5.4.2. 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 Figure 20. Development of the netsims proved challenging, insufficient detail was provided in the code development prompts

 doi:10.20944/preprints202512.0827.v1

on retry logic and and a range of approaches were implemented by the AI copilot. This was only identified at the end of the project, with insufficient time to redevelop the benchmarks, with two of four benchmarks excluded from this assessment. 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.

BLSTRS- BLS-Netsim

50ms +/-10ms + 2% packet loss									
40% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	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
120ms +/-30ms + 5% packet loss									
40% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	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

Figure 20. Results - BLSTRS BLS Netsim

5.5. Trusted Dealer Comparison

Figure 21 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 seconds 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.

 doi:10.20944/preprints202512.0827.v1

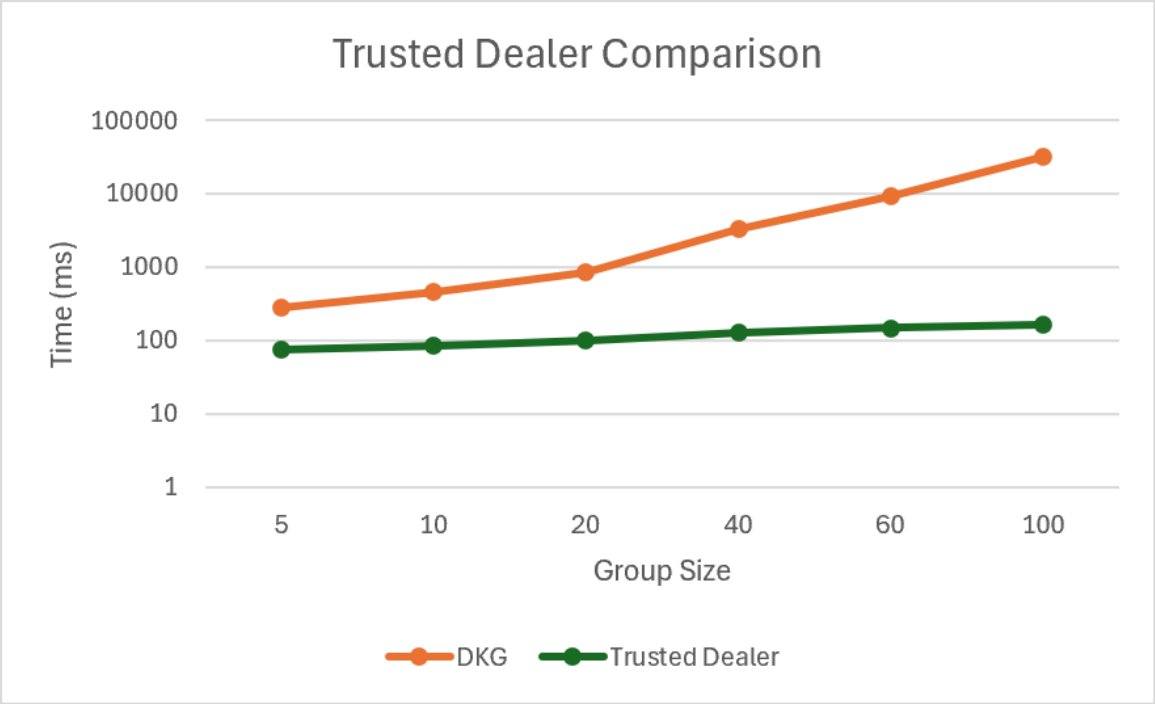

Figure 21. Results - Zcash Trusted Dealer Comparison

5.5.1. 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 100s or 1000s of signatures, during these 1000s 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 (50ms+/-10ms +2% packet loss), retry logic was fixed at 3 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 20ms 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.

 doi:10.20944/preprints202512.0827.v1

Figure 22. Results-Per Signature Performance

Figure 23 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.

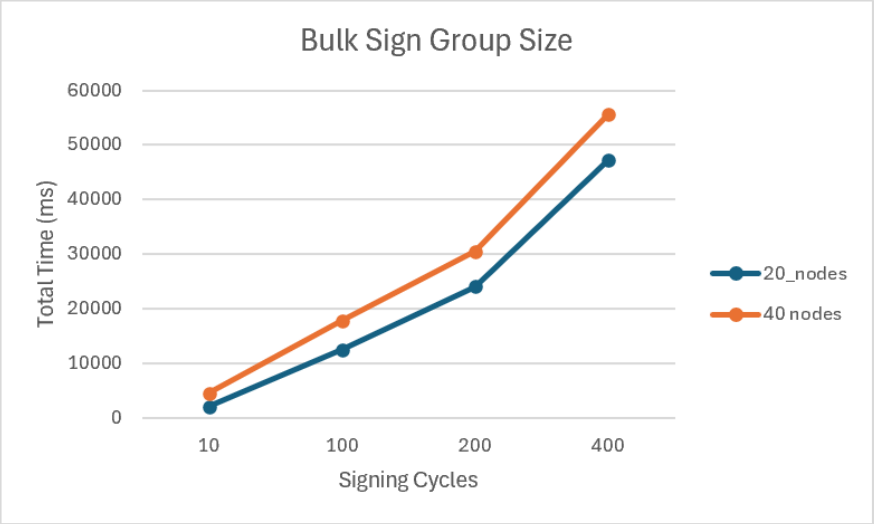

Figure 23. Results-Impact of Group Size

5.5.2. Threshold Signature Results Conclusion

These benchmarks have shown the validity of our methodology, the static benchmarks allowed the raw efficiency of each library to be assessed. While the netsims placed them in a realistic simulation, that allowed both protocol design and library efficiency to be evaluated. DKG was a clear constraint on performance for al 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 Chapter 4. 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 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

 doi:10.20944/preprints202512.0827.v1

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.

6. Conclusions

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).

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.

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.J.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.J.M and M.T.; supervision, W.J.B and R.J.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.

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.

 doi:10.20944/preprints202512.0827.v1

Appendix A. MLS Results

MLSpp - Full MLS Mode

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	33.32	213.44	25.74	212.01	5.637	5.924	6.405
	100	37.41	431.62	32.04	400.90	4.855	5.040	5.703

OpenMLS - Full Mode Results

Ciphersuite	Group Size	Add Member (ms)	Update Member (ms)	Remove Member (ms)	Process Update (ms)	Msg (100B) (ms)	Msg (1024B) (ms)	Msg (2048B) (ms)
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

 doi:10.20944/preprints202512.0827.v1

OpenMLS

Key Export Performance

Group Size	Median Time (ms)	CV (%)
10	0.0034	13.94
20	0.0050	10.24
50	0.0080	15.92
100	0.0132	13.32

Encryption and Decryption Performance

Message Size (bytes)	Encrypt Median (ms)	Encrypt CV (%)	Decrypt Median (ms)	Decrypt CV (%)
100	0.0004	8.33	0.0003	8.82
1024	0.0011	13.13	0.0010	8.02
2048	0.0019	10.42	0.0018	10.63

OpenMLS Total Time Performance

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

MLSpp

Key Export Performance

Group Size	Median Time (ms)	CV (%)
10	0.005683	5.16
20	0.00558	1.75
50	0.005655	3.02
100	0.005663	6.5

Encryption and Decryption Performance

Message Size (bytes)	Encrypt Median (ms)	Encrypt CV (%)	Decrypt Median (ms)	Decrypt CV (%)
100	0.0012	3.42	0.0011	2.58
1024	0.0019	3.1	0.0016	3.23
2048	0.0026	2.16	0.0021	4.52

MLSpp Total Time Performance

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

 doi:10.20944/preprints202512.0827.v1

Appendix B. Threshold Signature Results

Kryptology - FROST - DKG -Ed25519

33% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	22342.32	1.97	562.93	6.76	90.11	0.92	26535.31	2.53
66% Threshold		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	51.8	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	12114.81	1.39
100	66	42915.85	3.23	3471.01	3.61	150.893	4.18	54228.44	2.42

Kryptology - FROST -DKG- secp256k1

33% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	10077.84	2.53	284.52	10.91	549.06	40.49	11697.41	4.21
60	20	37907.52	14.50	1002.84	3.60	941.86	23.83	41344.78	4.37
100	33	172592.43	2.42	3396.30	1.80	993.40	30.52	173948.96	1.55
66%		**DKG**		**Signing**		**Verification**		**End to End**	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	18516.92	2.85	1270.55	3.82	579.55	21.83	19326.77	4.51
60	40	78514.81	1.83	5144.81	5.11	941.26	25.08	80340.13	5.45
100	66	315975.24	1.68	15975.60	1.71	905.33	26.08	347080.33	1.50

 doi:10.20944/preprints202512.0827.v1

Zcash Foundation - FROST - DKG - Ed25519

33% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median (ms)	CV (%)	Median (ms)	CV (%)	Median (µs)	CV (%)	Median (ms)	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	5.2
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	24357.32	0.5	28.91	6.64	115.21	50.9	24865.36	0.67
66% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median (ms)	CV (%)	Median (ms)	CV (%)	Median (µs)	CV (%)	Median (ms)	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	11089.14	1.11	37.98	5.33	114.27	17.97	10477.19	0.78
100	67	46831.99	1.38	101.66	3.5	110.42	379.14	47916.47	0.45

Bank Italia - FROST-DKG - Secp256k1

33% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	2.4
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	12978	10.61	291.07	88.88	63.9	7.16	12815	9.09
66% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	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	27044	4.13	1539.70	42.91	62.1	48.93	27871	4.26

 doi:10.20944/preprints202512.0827.v1

BLS-BLSTRS-DKG

33% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	35762.23	2.92	6.82	7.50	1551.24	2.28	35705.35	0.68
66% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	15128.73	1.38	8.48	8.05	1507.28	1.62	15297.59	0.75
100	67	70244.19	1.57	14.73	15.05	1513.93	2.39	70334.35	0.57

BLSTTC - BLS - DKG

33% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	CV (%)
5	2	11.22	4.18	1.60	5.54	1438.87	10.73	1451.69481	8.3
10	4	174.98	3.5	2.80	5.33	1340.23	5.81	1518.011255	8.03
20	7	2083.02	2.33	4.41	14.7	1363.79	4.42	3451.216934	4.52
40	14	34777.16	2.94	9.28	11.98	1367.71	5.53	36154.1427	4.14
60	20	155833.00	3.73	13.07	12.23	1375.75	6.79	157221.8173	5.1
100	34	1244927.00	3.72	6.36	13.03	1429.24	10.60	1246362.6	3.65
66% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (µs)	CV (%)	Median Time (ms)	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	123760.70	1.87	10.04	4.62	1373.18	6.63	4522.50	1.22
60	40	607845.40	0.15	10.43	17.58	1411.36	5.46	14990.82	0.89
100	67	-	-	-	-	-	-	-	-

 doi:10.20944/preprints202512.0827.v1

FROST-BankItalia Netsim - secp256k1

40% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)
5	2	160.4	22	129.7	na	0.2	na	350.4	13
10	4	343.3	15	216.9	na	0.2	na	500.5	12
20	8	457.4	5	272.4	na	0.1	na	722.1	6
40	16	1318	29	367.3	na	0.1	na	1685.7	26
120ms +/-30ms + 5% packet loss									
40% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)
5	2	na	na	na	na	na	na	na	na
10	4	542.5	34	415.2	na	0.2	na	995.5	27
20	8	1125	23	493	na	0.2	na	1625.9	17
40	16	1960.5	2	1008.4	na	0.1	na	2967.7	14

BLSTRS- BLS-Netsim

50ms +/-10ms + 2% packet loss									
40% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	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
120ms +/-30ms + 5% packet loss									
40% Threshold		DKG		Signing		Verification		End to End	
Nodes (n)	Threshold (t)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	CV (%)	Median Time (ms)	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

 doi:10.20944/preprints202512.0827.v1

FROST Bulk Signing Performance (Zcash Ed25519)

Nodes	Cycles	DKG Time (ms)	DKG CV (%)	Signing Time (ms)	Signing CV (%)	Verification Time (ms)	Verification 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
	100	784.44	11.2	11576.5	2.3	21.98	7.9	11627.94	1.97	12439.46
	200	784.44	12	23033.8	2	44.59	5.8	23234.25	1.38	24045.77
	400	815.31	12.5	45620	2	89.29	3.5	46391.39	0.91	47202.91
40	10	3055.57	9	1267.25	8.8	2.15	13.9	1392.77	4.48	4448.34
	100	2986.55	11.4	12693.5	3.1	22.03	7.2	14771.06	1.87	17826.63
	200	2896.95	8.4	25059.3	2.4	44.55	5.4	27367.17	1.64	30422.74
	400	2969.82	8.7	50335.9	2.3	89.26	3.5	52488.77	0.80	55544.34

FROST BankItalia Bulk Signing NetSim (secp256k1)

Nodes	Cycles	DKG Time (ms)	DKG CV (%)	Signing Time (ms)	Signing CV (%)	Verify (ms)	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	26137.8	2.2	13.6	10.4	26151.4	0.104	26629.9
20	200	460.9	0.9	52161.6	2.2	31.4	5.9	52193	0.0295	52653.9
20	400	546.5	0.5	103719	1.3	60.9	3.6	103780.1	0.009	104326.6
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	35745.3	1.9	14.9	3.4	35760.2	0.034	36664.2
40	200	926.8	1.3	71899.8	1.1	29.9	2.3	71929.7	0.0115	72856.5
40	400	814.5	0.5	147239	0.8	55.3	2.7	147294.4	0.00675	148108.9

BLSTRS Bulk Signing NetSim

Nodes	Cycles	DKG Time (ms)	DKG CV (%)	Signing Time (ms)	Signing CV (%)	Verify (ms)	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
	100	842.7	17.1	8589.1	4.9	5960.5	2.5	14549.6	17.27	15392.30
	200	794.6	4.4	17762.2	3.2	11559.4	1.6	29321.6	36.90	30116.20
	400	775.8	2.6	36397.4	2.2	23274.3	1.5	59671.7	76.92	60447.50
40	10	3631.1	5.7	1093.6	14.4	567.8	5.8	1661.4	0.46	5292.50
	100	3451.4	4.3	11329.6	2.8	5784	4.3	17113.6	4.96	20565.00
	200	3505.6	8.9	23041.7	3.1	11519.5	2.2	34561.2	9.86	38066.80
	400	3484.5	3.70	45584.9	2.2	23630.8	1.7	69215.7	19.86	72700.20

 doi:10.20944/preprints202512.0827.v1

Zcash DKG vs TD

Nodes	DKG Time (ms)	Trusted Dealer Time (ms)	Ratio
5	285.56	73.85	3.87
10	462.77	84.46	5.48
20	847.58	101.05	8.39
40	3,260.10	127	25.67
60	9,116.86	147.13	61.96
100	31,335.87	162.69	192.61

FROST BankItalia-DKG vs TD

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

BLSTRS BLS DKG vs TD

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 Transactions on Aerospace and Electronic Systems* **2023**, *59*, 8550–8564. https://doi.org/10.1109/TAES.2023.3307092.
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, Oct. 2018, pp. 5895–5900. https://doi.org/10.1109/IECON.2018.8591080.
3. El-Zawawy, M.A.; Brighente, A.; Conti, M. Authenticating Drone-Assisted Internet of Vehicles Using Elliptic Curve Cryptography and Blockchain. *IEEE Transactions on Network and Service Management* **2023**, *20*, 1775–1789. https://doi.org/10.1109/TNSM.2022.3217320.
4. Gupta, S.G.; Ghonge, M.M.; Jawandhiya, P.M. Review of Unmanned Aircraft System (UAS). *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* **2013**, *2*, 1645–1658.
5. Shachtman, N. Insurgents Intercepting Predator Video? No Problem. *Wired* **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 Aerospace and Electronic Systems Magazine* **2022**, *37*, 14–28. https://doi.org/10.1109/MAES.2022.3151308.
7. Abdalla, A.S.; Marojevic, V. Security Threats and Cellular Network Procedures for Unmanned Aircraft Systems: Challenges and Opportunities. *IEEE Communications Standards Magazine* **2022**, *6*, 104–111. https://doi.org/10.1109/MCOMSTD.2022.9973273.
8. Author's Name. Article Title. https://www.bbc.co.uk/news/articles/ckgn47e5qyno, Year of Publication. Accessed: 2025-08-28.
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, 2023. Issue 9.

 doi:10.20944/preprints202512.0827.v1

11. Civil Aviation Authority. CAP 2973: Cyber Security Guidance for Innovators. Technical Report 2973, 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, 2024. Submission in accordance with General Assembly Resolution 78/241.
13. Lamport, L.; Shostak, R.; Pease, M. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* **1982**, *4*, 382–401. https://doi.org/10.1145/357172.357176.
14. Shamir, A. How to Share a Secret. *Communications of the ACM* **1979**, *22*, 612–613. https://doi.org/10.1145/359168.359176.
15. Blakley, G.R. Safeguarding cryptographic keys. In Proceedings of the 1979 International Workshop on Managing Requirements Knowledge (MARK), Jun. 1979, pp. 313–318. https://doi.org/10.1109/MARK.1979.8817296.
16. Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In Proceedings of the 28th Annual Symposium on Foundations of Computer Science (SFCS 1987), Oct 1987, pp. 427–438. https://doi.org/10.1109/SFCS.1987.4.
17. Pedersen, T.P. A Threshold Cryptosystem without a Trusted Party. In Proceedings of the Advances in Cryptology — EUROCRYPT '91; Davies, D.W., Ed., Berlin, Heidelberg, 1991; pp. 522–526. https://doi.org/10.1007/3-540-46416-6_47.
18. Pedersen, T.P. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Proceedings of the Advances in Cryptology — CRYPTO '91; Feigenbaum, J., Ed., Berlin, Heidelberg, 1992; pp. 129–140. https://doi.org/10.1007/3-540-46766-1_9.
19. Administrator. Exclusive: British Navy to Pair F-35B Fighter Jets with Drones and Add Deck-Launched Missiles on Aircraft Carrier. https://armyrecognition.com/news/navy-news/2025/exclusive-british-navy-to-pair-f-35b-fighter-jets-with-drones-and-add-deck-launched-missiles-on-aircraft-carrier, 2025. Accessed: Jun. 11, 2025.
20. Chandramouli, A.; Choudhury, A.; Patra, A. A Survey on Perfectly Secure Verifiable Secret-sharing. *ACM Computing Surveys* **2022**, *54*, 1–36. https://doi.org/10.1145/3512344.
21. Gennaro, R.; Jarecki, S.; Krawczyk, H.; Rabin, T. Revisiting the Distributed Key Generation for Discrete-Log Based Cryptosystems. In Proceedings of the Topics in Cryptology — CT-RSA 2003. Springer, 2003, Vol. 2612, *Lecture Notes in Computer Science*, pp. 373–390. https://doi.org/10.1007/3-540-36563-X_25.
22. Ergezer, S.; Kinkelin, H.; Rezabek, F. A survey on Threshold Signature Schemes. In Proceedings of the Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM), Summer Semester 2020, 2020, pp. 49–54. https://doi.org/10.2313/NET-2020-11-1_10.
23. Sedghighadikolaei, K.; Yavuz, A.A. A Comprehensive Survey of Threshold Digital Signatures: NIST Standards, Post-Quantum Cryptography, Exotic Techniques, and Real-World Applications. *arXiv preprint arXiv:2311.05514* **2023**.
24. Schnorr, C.P. Efficient signature generation by smart cards. *J. Cryptology* **1991**, *4*, 161–174. https://doi.org/10.1007/BF00196725.
25. Stinson, D.R.; Strobl, R. Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates. In Proceedings of the Information Security and Privacy; Varadharajan, V.; Mu, Y., Eds., Berlin, Heidelberg, 2001; Vol. 2119, *Lecture Notes in Computer Science*, pp. 417–434. https://doi.org/10.1007/3-540-47719-5_33.
26. Boneh, D.; Lynn, B.; Shacham, H. Short Signatures from the Weil Pairing. In Proceedings of the Advances in Cryptology – ASIACRYPT 2001; Boyd, C., Ed. Springer, Berlin, Heidelberg, 2001, Vol. 2248, *Lecture Notes in Computer Science*, pp. 514–532. https://doi.org/10.1007/3-540-45682-1_30.
27. Boneh, D.; Gentry, C.; Lynn, B.; Shacham, H. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In Proceedings of the Advances in Cryptology – EUROCRYPT 2003; Biham, E., Ed. Springer, Berlin, Heidelberg, 2003, Vol. 2656, *Lecture Notes in Computer Science*, pp. 416–432. https://doi.org/10.1007/3-540-39200-9_26.
28. Diffie, W.; Hellman, M. New Directions in Cryptography. *IEEE Transactions on Information Theory* **1976**, *IT-22*, 644–654.
29. Steiner, M.; Tsudik, G.; Waidner, M. Diffie-Hellman Key Distribution Extended to Group Communication. In Proceedings of the Proceedings of the 3rd ACM Conference on Computer and Communications Security, New Delhi, India, 1996; CCS '96, pp. 31–37. https://doi.org/10.1145/238168.238182.

 doi:10.20944/preprints202512.0827.v1

30. Cohn-Gordon, K.; Cremers, C.; Garratt, L.; Millican, J.; Milner, K. On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees. In Proceedings of the Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, Oct 2018; CCS '18, pp. 1802–1819. https://doi.org/10.1145/3243734.3243747.
31. 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. hal-02425247.
32. Castro, M.; Liskov, B. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems* **2002**, *20*, 398–461. https://doi.org/10.1145/571637.571640.
33. Bondar, K. How Ukraine's Operation "Spider's Web" Redefines Asymmetric Warfare. https://www.csis.org/analysis/how-ukraines-spider-web-operation-redefines-asymmetric-warfare, 2025. Accessed: 2024-07-23.
34. ISW Press. Russian Force Generation and Technological Adaptations Update: June 27, 2025. https://www.understandingwar.org/backgrounder/russian-force-generation-and-technological-adaptations-update-june-27-2025, 2025. Accessed: 2024-07-23.
35. Noguchi, T.; Nakagawa, M.; Yoshida, M.; Ramonet, A.G. A Secure Secret Key-Sharing System for Resource-Constrained IoT Devices Using MQTT. In Proceedings of the 2022 24th International Conference on Advanced Communication Technology (ICACT), 2022, pp. 147–153. https://doi.org/10.23919/ICACT53585.2022.9728781.
36. Tan, H.; Zheng, W.; Vijayakumar, P. Secure and Efficient Authenticated Key Management Scheme for UAV-Assisted Infrastructure-Less IoVs. *IEEE Transactions on Intelligent Transportation Systems* **2023**, *24*, 6389–6400. https://doi.org/10.1109/TITS.2023.3252082.
37. 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 Proceedings of the 2025 4th International Conference on Cyber Security, Artificial Intelligence and the Digital Economy, Kuala Lumpur, Malaysia, 2025; pp. 125–131. https://doi.org/10.1145/3729706.3729725.
38. Yang, Z.; Wang, Z.; Qiu, F.; Li, F. A group key agreement protocol based on ECDH and short signature. *Journal of Information Security and Applications* **2023**, *72*, 103388. https://doi.org/10.1016/j.jisa.2022.103388.
39. Leon, A.; Britt, C.J. UXS Authentication and Key Exchange Requirements for Multidomain Operation and Joint Interoperability. PhD thesis, Naval Postgraduate School, Monterey, CA, 2022. Doctoral dissertation.
40. 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), 2023.
41. Marstrander, E. Use of Messaging Layer Security in a Military UAV Swarm. Master's thesis, Norwegian University of Science and Technology (NTNU), 2023.
42. 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, California, 2019. https://doi.org/10.2514/6.2019-0689.
43. 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., Berlin, Heidelberg, 2003; Vol. 2567, *Lecture Notes in Computer Science*, pp. 31–46. https://doi.org/10.1007/3-540-36288-6_3.
44. Gennaro, R.; Jarecki, S.; Krawczyk, H.; Rabin, T. Secure Applications of Pedersen's Distributed Key Generation Protocol. In *Topics in Cryptology — CT-RSA 2003*; Joye, M., Ed.; Springer, Berlin, Heidelberg, 2003; Vol. 2612, *Lecture Notes in Computer Science*, pp. 152–170. https://doi.org/10.1007/3-540-36563-X_26.
45. Komlo, C.; Goldberg, I. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. Cryptology ePrint Archive, Paper 2020/852, 2020.
46. Al-Tawil, I.; Al-Tawil, S.M.; Al-Tawil, M.I. Benchmarking Cryptographic Protocols for IoT Malware Defence. In Proceedings of the International Conference on Information and Communication Technology and Baha'i Faith (ICTBF). IEEE, 2021, pp. 1–6. https://doi.org/10.1109/ICTBF52762.2021.9472304.
47. Geisler, M. Digital signatures in wireless sensor networks. Master's thesis, Aalborg University, 2008.
48. Leppänen, P. Performance measurement of cryptographic protocols. Master's thesis, University of Oulu, 2014.
49. Group, I.M.W. The Messaging Layer Security (MLS) Protocol. RFC 9420, 2023.
50. MLS Working Group. MLS Implementations. GitHub, 2024. Accessed: August 24, 2025.

 doi:10.20944/preprints202512.0827.v1

51. Roch, Paul. Dissertation. GitHub, 2024. Accessed: August 24, 2025.
52. Kiefer, F. Post-Quantum OpenMLS **2024**. Accessed: 2025-07-15.