CEAS Aeronautical Journal
https://doi.org/10.1007/s13272-025-00806-5

ORIGINAL PAPER

Agile, post-quantum secure cryptography in avionics

Karolin Varner[2,3] · Wanja Zaeske[1,3] · Sven Friedrich[1] · Aaron Kaiser[2] · Alice Bowman[3]

Received: 12 April 2024 / Revised: 17 December 2024 / Accepted: 9 January 2025

Abstract
To introduce a post-quantum-secure encryption scheme specifically for use in flight-computers, we used avionics' module-isolation methods to wrap a recent encryption standard (HPKE-Hybrid Public Key Encryption) within a software partition. This solution proposes an upgrade to HPKE, using quantum-resistant ciphers (Kyber/ML-KEM and Dilithium/ML-DSA) redundantly alongside well-established ciphers, to achieve post-quantum security. Because cryptographic technology can suddenly become obsolete as attacks become more sophisticated, "crypto-agility"—the ability to swiftly replace ciphers—represents the key challenge to deployment of software like ours. Partitioning is a crucial method for establishing such agility, as it enables the replacement of compromised software without affecting software on other partitions, greatly simplifying the certification process necessary in an avionics environment. Our performance measurements (Sect. 5) provide initial evidence that both the memory and cpu performance characteristics of this solution are suitable for deployment in flight-computers. Performance measurements show a memory use of 5 MB of RAM and under 200 KB of stack usage for encryption, compared to a baseline implementation without any encryption; decryption is much more lightweight (under 300 KB RAM overhead, under 100 KB of stack requirement overhead). Generally, the post-quantum algorithms benchmarked where faster than their pre-quantum alternatives; due to the use of hybrid security this leads to a performance overhead of just about 90% compared to the pre-quantum only variant. The implementations benchmarked are optimized for CPU-performance and alternative, lower quality implementations showed much more modest memory requirements, leading us to conclude that there is much room for optimization, targeting use-case specific tradeoffs between memory use and performance.

Keywords Avionics · Crypto-agility · Post-quantum cryptography · Robust Combiners · HPKE · ML-KEM · ML-DSA

1 Introduction

Both airlocks and aircraft communication systems require careful maintenance. However, whilst a broken airlock may be easily located in the course of a visual inspection, a broken security system for aircraft communications will appear no different to an intact one. Although the creation of either system requires years of work from dedicated engineers to evaluate, test, assess, and optimise each constituent part, the fruit of that labour looks rather different. In place of a material construction of plastic and aluminium, we find ourselves presented with an abstract construction of information theory: mathematical consideration presented in code.

Given the intangible nature of such communications security systems, it is perhaps unsurprising that their state-of-the-art manifestations are conspicuously absent from the broader effort to ensure the safety of aircraft. Whereas the principles justifying the safety of an airlock are bound by consistent laws of physics, the threats posed by malicious software attacks remain amorphous in their nature.

Irrespective of their ever-changing form, threats to security remain severe in both outcome and scope. An ineffective security scan may allow a bad actor to board a plane with a weapon, for example. An attack on Aircraft Communications Addressing and Reporting System (ACARS), potentially from a hostile nation state, could guide two aircraft into a collision. Breaking that critical connection between ground control and pilot, and especially inserting an undiscovered participant in the middle of said connection, could

✉ Wanja Zaeske
wanja.zaeske@dlr.de

[1] Department Safety Critical Systems and Systems Engineering, Institute of Flight Systems, German Aerospace Center (DLR), Lilienthalplatz 7, Braunschweig, Germany

[2] Max Planck Institute for Security and Privacy, Universitätsstraße 140, 44799 Bochum, Germany

[3] Rosenpass e.V., Postfach 3212, 30032 Hannover, Germany

Published online: 12 May 2025

allow for unspeakable tragedies should aeroplane pilots find themselves unable to co-ordinate with neither the ground nor one another.

A successful Denial of Service attack on communications systems could, in extreme cases, ground a significant amount of air traffic, causing economic wreckage whilst undermining public trust in avionics as a whole. Both aircraft voice radios and ACARS are both unencrypted and unauthenticated, despite the ready availability of the technologies developed by cryptographers to secure them.

High public trust in avionics, and aviation in general, is the result of of this sector's insistence on developing and improving the safety of its planes. Nonetheless, despite the availability of encryption, only a small amount of communication in avionics employs cryptography. For example, Smith, Moser, Strohmeier, Lenders, and Martinovic claim that 99% of ACARS data traffic is sent in plain text [1], Schäfer, Lenders, and Martinovic elaborate on the widely used Automatic Dependent Surveillance-Broadcast (ADS-B) which employs no cryptography [2].

If we are to prevent the disastrous scenarios described above, all aircraft communication should be encrypted or signed. We propose, that the most effective approach to achieving this is by fostering closer collaboration between researchers in avionics and cryptographers. Moreover, the techniques studied in cryptography are well-suited for application to avionics' unique challenges, such as guarding redundant flight computers against even extreme error cases.

There is good reason for the approach to the development of avionic protocols to be one of conservative and considered development. Very few technologies employed in avionics are subject to rapid obsolescence, and many solutions of yesteryear work just as well as ever. The scientific challenges faced by avionics engineers are less capricious than that of the ever-evolving threat environment faced by cryptographers, and tend towards being solved reliably rather than absolutely, with risk being managed as a percentage, rather than a binary. The absolute security often favoured by cryptographers would see an encrypted message discarded entirely, rather than see it accepted with minimal errors such as bit flips.

When offered the choice between availability and integrity, cryptographers will almost always choose integrity. This is because they model the world as a relationship between honest parties and attackers, with technological constraints, random errors, and bit flips conceptually belonging to this attacker and to be thwarted in the same way as any other attack. In encryption, this is realised by discarding any messages containing errors, a sensible approach when the message is a communication with a website that can be easily re-transmitted. Of course, it lends to more undesirable outcomes if, for example, the data being transmitted is part of pilot-to-tower radio communication during an emergency landing.

The deployment of cryptography within avionics poses an additional challenge: development cycles in avionics are slow, and thus upcoming standards must foresee and address the potential needs of avionics systems forty years ahead, not only those of today. Quantum computers, however, are threatening today's cryptosystems and, while cryptographers have been working hard to establish cryptosystems to counter this threat, those systems have only recently become standardised. Any cryptography standard within avionics, in pursuit of both reliability and safety, must maintain the tried-and-true techniques of classical, that is "pre-quantum", cryptography. Fortunately, with this issue having arisen in many fields, the well-researched techniques of hybrid cryptography (redundant cryptosystems) and crypto-agility (methods to quickly migrate between cryptographic techniques) are readily applicable.

Cryptographic systems are a sequence of choices, usually ones made under the assumption that the data transmitted is so benign that a loss of connectivity is inconsequential. In most applications, that assumption is a sound one. Readdressing that balance, between availability and integrity, is made more complex by the catastrophic nature of any cryptographic failure.[1] Finding the right compromise requires careful collaboration between domain experts, who understand the needs of their environment, and cryptographers, who can communicate the implications of said compromise.

1.1 How to read this paper

This paper is split into editorial, background, and novel results sections. The most concise version of our results is to be found in Our Contribution and Conclusion.

As this work is the synthesis of contributions by avionics researchers and cryptographers, this paper attempts to ensure the results are accessible to researchers from both disciplines by providing an extended section for background information: "Avionics & Cryptography" provides an overview of the techniques used in both fields. Readers familiar with cryptography might want to read subsections "Avionics Software Engineering" and "Comparing Avionics and Cryptography" only; readers familiar with avionics but not cryptography may skip those sections entirely.

Both the Introduction and Conclusion are written in an editorial style, to include readers who would like to take a broader, hollistic view on the subject.

"Our Contribution" provides an overview of the novel results in this paper. Post-quantum security for HPKE

[1] There are cases of cryptographic constructions where even seemingly minimal leakage of information can present as an open door for attackers to fully recover the most important secret information.

discusses the cryptographic aspect of our work in-depth, and Integrating HPKE in an ARINC 653 partition does the same for the avionics-engineering component. In the section Evaluation: Performance and Memory Overhead, we analyse the efficiency and performance characteristics of our solution.

2 Avionics and cryptography

2.1 Avionics software engineering

Beginning in the 70's, and following into the modern day, ever increasing numbers of aircraft functions are, at least partially, software-defined. As a response to the growing risks posed by software failures, disparate software safety regulations were standardised within the DO-178 in 1981. Its newest iteration takes a holistic approach to the software lifecycle, including the planning, development, and integral [3] processes.

Aviation agencies across the globe certify the safety, and therefore air worthiness, of avionics software according to the objectives set out in this standard. It is able to distinguish between safety-critical components and less important ones, evaluating the severity of their fail-cases and centrality to the craft itself.

Integral to DO-178C's ability to separate software concerns is its concept, and implementation, of partitioning. That is, to break software into well-isolated pieces that are easy to contain in the event of failure. As long as a fault in one partition is contained to said partition, its neighbouring software components are able to function as normal, despite running on the same hardware. This achieves numerous safety goals, including the concession to human frailty that, even with the most rigorous testing available, software components may contain uncaught flaws.

Another benefit of the partitioning concept lies within the broader, and often onerous, certification process. Firstly, by assigning varying safety levels to separate partitions, less essential components can be subjected to less rigorous testing [3], lowering the barriers to the development of such inessential software [3, ch. 2.4.1]. Secondly, and crucially to this paper, a software component can be designated, via the issue of a reusable software component acceptance letter [4], practically a "plug and play" component, thereby reducing the burden of certification to its integration within a specific system.

To facilitate a common platform, ARINC 653 describes an Application Programming Interfaces (API) for partitioning avionic hypervisors [5]. Many implementations of this API exist, allowing for software to be written independently of a specific hypervisor. However, ARINC 653 provisions for some degree of freedom in the API. We developed a653rs,[2] a Rust port of the ARINC 653 API that further exacts these. To enable fast prototyping, we further developed a653rs-linux[3] which provides an ARINC 653 like environment on top of any recent Linux based operating system [6].

2.2 Certification in avionics

While DO-178C's objectives and guidelines are more concerned with the broader development of avionics software, DO-297 focuses said software's integration into the aircraft themselves. Modularity is a key property in managing the complexity of building aircraft, with the predominant design philosophy embracing Integrated Module Avionics (IMA), a modular approach towards avionics. However, a system integrating many modules depends on the correct interactions between said modules to function properly. Both the individual behaviour of modules, as well as their interplay, must be accounted for.

To better separate the scope of various elements in the system composition, we shall outline some terminology below that allows us to be more specific regarding the processes necessary for acceptance and certification.

Aircraft Function A capability that is provided by hardware and/or software on an aircraft. For example flight control, autopilot, fuel management, flight instruments etc.

Application "Software and/or application-specific hardware with a defined set of interfaces that, when integrated with a platform, performs a function" [4]

Component "A self-contained hardware part, software part, database, or combination thereof that is configuration controlled. A component does not provide an aircraft function by itself" [4]

Module One or multiple components (hardware and/or software) that provide resources to the IMA-hosted applications [4].

The actual certification process for an aircraft involves four consecutive steps:

1) Module acceptance
2) Application acceptance
3) System acceptance
4) Aircraft integration

[2] https://github.com/DLR-FT/a653rs.

[3] https://github.com/DLR-FT/a653rs-linux.

Two additional steps describe iteration on modules and applications:

1) Change of modules or applications
2) Reuse of modules or applications [4]

The first two steps (module and application acceptance) are focused more on individual elements in the system. The concept of Reusable Software Components (RSCs), introduced in AC 20–148, enables some certification credit to be reused [7]. While limited to software, this enables cost savings upon certification when employing a proven software component into a new type of aircraft. As Reusable Software Component (RSC) are not viable for the system acceptance and aircraft integration tasks, the certifiably correct integration of a RSC into the system/aircraft context still remains open work for future aircraft type certifications.

2.3 Cryptography

Cryptography, at its core, is the study of methods to secure communications and information processing operations using mathematics. Possession of a piece of information, is the difference between a successful attack and business as usual. The classic application of cryptography is in encryption: securing a message against modification, forgery, and surveillance.

While many think only of encryption when they hear "cryptography", encryption is but one tool in a cryptographers' toolkit. Cryptography is also the study of essential techniques such as: performing computations on secret data [8], mathematically certifying bureaucratic processes [9], and introducing redundancy to computations to secure them against faults and tampering [10].

Often, the functional goal of the problem under study is trivial: transmitting a message is not hard, one simply hands it to the recipient. Counting a vote or calculating statistics also do not present as difficult problems, at least mathematically speaking: the methods used may be intricate, but they have long since been examined and are considered solved problems.

Cryptographers instead focus on the attempt to construct systems in which cheating, sabotage, and manipulation are impossible. In a secure voting system, voters would collaborate to count the vote to ensure a correct outcome, unless too many devices are compromised. In a secure joint scientific study, multiple parties could use cryptography to calculate the aggregate results, without having to disclose their own data individually.

By modelling problems encountered in the real world using mathematical terms, cryptographers seek to create an appropriate, probabilistic definition of security, develop new ciphers, and use mathematical proofs to certify the security of their systems.

Practical cryptographers then work to apply these mathematical findings to real-world systems. They are tasked with connecting abstract and simplified models, combining the complexity of hardware with the psychological, sociological, and legal mechanisms governing human behaviour.

Whereas a mathematical cryptographer may define a model in which some information—a key—is kept secret, practical cryptographers are tasked with determining whether the system actually keeps that key secret in real-world use.

Human ingenuity makes metal fly, and that same creative ingenuity is put to task subverting security systems. Over the decades, security analysts have found various forms of information *leakage*, resembling the sorts of stories typically found in a spy thriller: bags of potato chips used as makeshift microphones [11], fingerprints recreated from photographs, electrical traces on a circuit board turned into antennas [12], and the fluctuations of a computer's power light used to extract cryptographic keys [13].

Engaging in this arms race requires both a full-system perspective and a cross-discipline approach. The systems deployed must be well understood, especially their interactions, under even the most unlikely of circumstances. If a modulating power light can give away a secret key, one cannot confine oneself to the study of just the key alone. The perspective of experts in a broad range of fields is necessary to keep the potential security implications of one, quietly blinking, LED power light in mind.

It should be no surprise that open standards, open science, open implementations, and open communication form the most important tools in a discipline focused on preserving secrets:

Cryptography is concerned with the creation of systems that remain secure, even in the presence of the most well-informed, sophisticated, malicious attacker possible. Cooperation, collaboration, and openness are central to cryptography, because we need all the help we can get to beat the most powerful attackers.

2.4 The basics of encryption

While the basic components of classical, i.e. pre-quantum, encryption have not changed much in the previous decades, the systems built from these components have evolved drastically. Many of these improvements focused on improving usability, allowing relatively inexperienced developers to deploy secure cryptography within their applications.

Fundamentally, what is expected from encryption is relatively clear. Interception of an encrypted message should not reveal its content or allow an attacker to modify the message (*confidentiality* and *integrity*). An attacker should not be able to send a message in the name of another (*authenticity*), and

neither should an attacker be able to prevent transmission of the message in the first place (*availability*). Availability proves hard to guarantee: There is no beating an attacker trying to jam your signal if they have the bigger antenna, so availability often takes a back seat to the other properties.

Engineering a system to achieve these goals in a particular environment is much harder as understanding which requirements to focus on demands a good understanding of the particular field. Cryptography is not just a mathematical and technical discipline; on a philosophical level, cryptographers have to determine what constitutes security as a concept. This process is often driven by researchers analysing attacks against a system, and then working backwards to create the appropriate security concepts that integrate the realities of deployment.

In this section, we explore the questions in need of answers before one can begin to develop an encryption system. What type of communication should be supported? How many parties are participating? Are these parties exchanging messages in a live session like a chat or a conversation, or is the communication akin to letters being exchanged? How safety-critical are the messages? Is it more important not to lose messages, or is it more important to prevent tampering?

Different types of requirements to inquire about before engineering a cryptographic system.

- Interactive or Non-interactive?
- One-way or Bidirectional?
- One-to-one, One-to-many, or Many-to-many?
- Integrity or Availability focused?
- In sequence or Any Order?
- Authenticated or Arbitrary sender?
- Should it be possible to audit the communication later?
- Is Anonymity required?

Other questions have been decisively answered by past cryptographic research. Unfortunately, new systems making inadvisable choices are still commonly being deployed. Whenever a new encryption system is specified, the consortium should be aware of what needs to be done to achieve practical security. The following sections discuss some of these needs.

Even in settings where availability is more important than integrity, some form of *integrity protection* should be used. Some standards, such as the Project 25 radio standard [14], forgo integrity protection entirely, rendering protocols vulnerable to a broad range of attacks. Ciphers that strike a better balance between integrity protection and availability are currently in development [15].

Cryptographic systems generally feature a *nonce*, a number used once, to ensure that exactly the same message is never transmitted twice. Even if the plaintext is the same, the nonce will be different. Nonces are often transmitted as part of the ciphertext, adding a few bytes of overhead. It can be tempting to try and improve the efficiency of a cipher by shortening the nonce, limiting the number of messages that can be supported. Unless done with exceptional caution, this is unsafe. The security of the underlying cipher has a hard requirement on no key/nonce combination *ever* being used twice and there are statistical effects, such as the birthday problem [16], that complicate the situation. Especially when symmetric keys are used over a long period of time or by multiple computers, a nonce of at least twenty-four bytes should be used. AES-GCM [17], one of the most commonly used ciphers, only supports a twelve-byte nonce and a 64GiB message size. It can be used securely, but meeting engineering requirements presents its own challenge.

This is partially why *asymmetric cryptography* should be used, whereby new symmetric keys are generated for every interaction. This type of encryption separates the keys into public keys and private keys; public keys can be used to encrypt messages and validate signatures, private keys can be used to decrypt messages and generate signatures. Consider Alice, a journalist, trying to allow a room full of potential sources to contact her in secret. With symmetric cryptography, each potential source would have to write a key into an envelope and pass it to Alice whilst making sure, in the process, that nobody can observe them writing down their key. With public key cryptography, Alice can simply write her key on a sign. Most modern encryption technologies rely on asymmetric cryptography as simplifying the key distribution process saves a massive amount of money when compared with the additional computational cost.

The ciphers used should be *post-quantum secure*, i.e. they should be resistant to cryptographic attacks from quantum computers.

There are also specific attacks that need to be guarded against, such as *side-channel attacks*, the aforementioned information leakage. The established standard is that systems should, at the very least, not leak any information as a result of timing behaviour.

Finally, the deployment must provide *cryptographic agility*. A breakthrough in cryptographic attacks can quickly and violently render systems vulnerable to attacks. Upgrade-procedures must be developed, and practised, to quickly recover security in such situations High-security systems can also be designed with redundancy in mind; using *robust combiners* [18] yields ciphers that remain secure, even if one component is broken, buying valuable time to perform necessary software upgrades.

Some minimum features for practical, secure encryption systems.

- Provide Integrity Protection.
- Support a Large Nonce.
- Provide Asymmetric Keys.

- Implement Timing-Side-Channel Resistance.
- Ensure Post-Quantum security.
- Practice Cryptographic Agility.

2.5 Engineering encryption systems: a checklist

There is no fixed set of techniques that can be used to create a secure, cryptographic system. Nonetheless, there is a set of best practices that can be followed to avoid common pitfalls. Some of these concepts will be introduced later in the paper.

Checklist of some steps that can be taken to improve the security of encryption systems.

System architecture

- Use open standards
- Provide cryptographic agility
- Account for changing algorithms

Cryptographic algorithms

- Use public development processes and open science
- Use standardised ciphers
- Require a proof of security
- Use well-established ciphers
- Additionally use post-quantum secure asymmetric ciphers

Implementation

- Use open-source software
- Use libraries instead of new implementations
- Provide the ability to perform software updates
- Sign software-updates
- Use memory-safe programming languages
- Erase secrets after use
- Check security against timing side-channels

2.6 Off-the-shelf encryption systems

When applying encryption to avionics, we should start with an off-the-shelf system, thereby building on the immense amount of research that has been invested in the design of these solutions. While they have not been purpose-built for use in aircraft, they are nonetheless well suited to jump-start the use of encryption in this field. Once the most immediate needs are well addressed, cryptographers and avionics specialists can collaborate to ensure avionics-specific needs are met in future versions of the system.

Some state-of-the art, standardised encryption systems. None of these provides post-quantum security by default.

HPKE [19] "Hybrid Public Key Encryption" Minimal, decentralised, asynchronous encryption standard. This could be built upon if none of the complex standards is suitable. Providing post-quantum security for HPKE is one result from this paper.

TLS 1.3 [20] "Transport Layer Security" Point-to-point, live communication over reliable and unreliable channels; certificate-based key distribution.

MLS [21] "Messaging Layer Security" Chat-like encryption of small and large data; asynchronous, multiple participants; federated—servers with good connectivity are required, but participants can run their own server. It could be used for applications like sensor-networks with intermittent connectivity.

TLS, *Transport Layer Security* [22], previously known as SSL, Secure Socket Layer, is one of the longest-standing encryption protocols. TLS provides support for certificates [23]—identification documents issued by a central authority for protocol participants. SSL is plagued by a number of security issues. One egregious issue is the fact that, even after an upgrade, deployments may remain vulnerable due to *downgrade attacks* [24, 25] as a result of its *cipher suite negotiation* feature—the ability to switch to an older, less secure, protocol version—for backwards compatibility. This feature must be actively disabled when using TLS in a secure system. An attacker can use this to lie to each end, pretending that secure cipher-suites are not available. *Transport Layer Security 1.3* is the first version of SSL/TLS that was developed with provable security in mind and removed some of these insecure features. Deployment of post-quantum secure cryptography in TLS [26–30] remains a work in progress.

The *Noise Protocol Framework* [31] is the result of a scientific study investigating the variety of key-exchanges that could be performed using the *Elliptic Curve Diffie-Hellman* [32] operation and pre-shared keys. Its implementations can serve as a simpler alternative to SSL/TLS, as well as an asymmetric encryption device similar to HPKE. Its simplicity is its advantage, as it reduces effort for cryptanalysts in a manner similar to how smaller applications are easier to certify. The Noise Protocol cannot be migrated to the post-quantum world without major modification. Thus, while it does not represent the future of cryptography, it certainly represents one of its key milestones.

The Signal protocol [33] is the protocol that underpins the security of the *Signal* messenger. Not a standard, but widely successful nonetheless. It found use in other platforms, such as the WhatsApp messenger. Later versions of the protocol were among the first to introduce advanced security features such as: group-messaging, *post-compromise-security*, and anonymous communication. *MLS* [21] is a recent standard-protocol; it pursues similar goals to the Signal protocol while

being developed as a standard. MLS does not currently support post-quantum security, but it was designed with a simple migration-path in mind.

While the motivating use case for chat protocols is human conversation, the possibilities endowed by these technologies are, in fact, broader. These protocols feature multiple participants broadcasting messages, including large multimedia files, to all participants in a chatroom securely. The participants are sometimes online, sometimes offline, and new chat rooms can be created on demand. The ability to establish security on sight helps ensure the security of future communication. This could be humans chatting. This could also be a network of sensors.

Finally, *Hybrid Public Key Encryption* [19] is an encryption standard designed for single-recipient, optionally authenticated encryption. In some ways, this protocol is designed to provide a tiny, baseline specification of asymmetric encryption-capabilities. This protocol was also created with post-quantum security in mind, though does not provide it in its initial variant.

2.7 Post-quantum cryptography

Quantum computing, while often hailed as the next major breakthrough in general computation, merely shows potential to accelerate the calculation of a select few problems from computer science. Some of those problems, whose inefficiency asymmetric cryptography relies upon, could be solved by quantum computers once they are built. Quantum computers differ fundamentally from the computers we are using today, as they employ findings from theoretical physics to create devices operating on a different type of information with a different set of operations. As yet, quantum computers have never been used in practice. While their mode of operation has some advantages, they pose many inefficiencies when compared to classical computers [34]. Some problems are likely to remain beyond the capabilities of both classical and quantum computers.

That some problems are beyond the ability of quantum computers to solve efficiently offers some genuine reprieve, as this has enabled cryptographers to research and develop cryptosystems that remain immune to the threat posed by quantum computers. This field is called *post-quantum cryptography*. Avionics must migrate to post-quantum cryptography as, once sufficiently large quantum computers are built, they will be immediately able to attack cryptosystems currently in use. Even prior to that, quantum computing represents a threat to the security of private and secret information, as attackers can simply store any encrypted data they harvest today, then wait until a quantum computer becomes available to decrypt it. Due to the long development times, certification delay and decades of operation, beginning this migration is a particularly urgent need for avionics.

The process to migrate to these new cryptosystems began in earnest a decade ago, and has met practical use with the start of the *NIST post-quantum cryptography competition*. In round three of the standardisation process, suitable ciphers[4] were selected for use [35]. These can now be employed on classical computers without major constraints. The authors of this papers' decades-old laptops easily run these algorithms.

The McEliece cryptosystem was developed in 1978 [36], though its quantum-resistant properties were discovered later. While widely believed to be secure, its modern incarnation, dubbed "Classic McEliece" [37], was not selected by NIST for standardisation, likely due to its large public-keys sizing in the hundreds of kilobytes. Instead, NIST selected Kyber [38] as the first post-quantum cipher to be standardised, along with the post-quantum signature schemes Dilithium [39], FALCON [40], and Sphincs+ [41]. During the NIST standardization of Kyber and Dilithium, they were renamed to Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM) and Module-Lattice-Based Digital Signature Standard (ML-DSA), respectively. When referring to ML-KEM and ML-DSA, we refer to the FIPS standards 203 [42] for ML-KEM and 204 [43] for ML-DSA.

2.8 Migrating encryption protocols to post-quantum security

As symmetric encryption is significantly faster than asymmetric encryption, protocols tend to be separated into both asymmetric components, that establish a shared-key, and symmetric components using said key to transmit the payload. Over time, cryptographers have began to rely on standard-components when designing cryptographic protocols. Generally speaking, a cryptographic protocol is secure if its ingredients are secure. Likewise, a protocol is post-quantum secure if its components fit that description.

Engineering constraints, such as key-sizes and performance budget, aside, the reliance on standard components should allow for a straightforward migration to post-quantum security. Simply migrate each component to post-quantum security. Unfortunately, migrating the most popular asymmetric components to post-quantum security has proven elusive to cryptographers for some time now. This leaves cryptographers little choice other than to painstakingly migrate each protocol to use ingredients with a different interface.

[4] Technically these asymmetric post-quantum schemes are not ciphers. They are called KEMs, "Key Encapsulation Mechanisms". They transfer a randomly chosen key and are not suitable for general encryption, but are purpose-built to construct general encryption systems when used together with other components. This paper subsumes these post-quantum key-transferral techniques under the term cipher for readability.

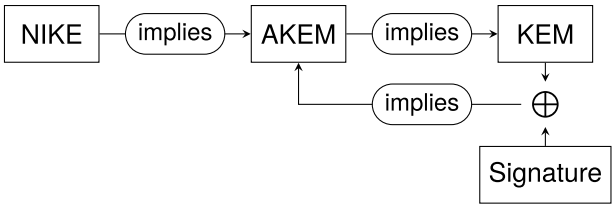

Fig. 1 NIKE is the strongest asymmetric interface; a NIKE can do anything a KEM or an AKEM can do. The novel AKEMs are closer to NIKEs than KEMs. An AKEM can be built from a NIKE or from a KEM and a Signature scheme. "Implies" in this context is a technical term from mathematics; you can read "NIKE implies AKEM" as: "An AKEM can be constructed by using just a NIKE"

Recently, the HPKE standard proposed a new device to aid cryptographers in their quest to migrate these protocols. It was clear, to most cryptographers, that an asymmetric key transferral (i.e. KEMs) could be well combined with signatures. Essentially, KEMs would provide secrecy, Signatures would provide authenticity, and the symmetric portions would ensure integrity. During the development of HPKE, its authors concluded that this mechanism could be used to create a primitive more akin to the interface (Fig. 1) used before the post-quantum migration. The *authenticated KEM* was born.

Authenticated KEMs ease protocol designers' work, as they are closer to the interface most commonly used in the pre-quantum world.

Some of the ingredients of a cryptographic protocol. Since NIKEs do not exist in the post-quantum world, the cryptographic aspect of migrating a protocol to post-quantum security is largely about replacing NIKEs with KEMs, AKEMs and Signatures. See Appendix 3 for a description of these interfaces.

AEAD "Authenticated Encryption w. Associated Data" Post-quantum variants: ✓ Standard symmetric encryption interface. Provides secrecy, authenticity, integrity, and has the ability to certify additional data from the communication context that does not need to be transmitted, such as the name of the sender.

NIKE "Non Interactive Key Exchange" Post-Quantum variants: × The current gold-standard interface for asymmetric encryption—i.e. transferring a symmetric key—in classical cryptography. This is considered "non-interactive" as no data, in addition to the public keys, has to be transmitted in order to exchange a symmetric key and, because the process provides implicit authenticity, requires no signatures. Both participants simply call the NIKE function with their own private key and the other participant's public key, respectively. Both produce the same shared key in the process. The available implementations were highly efficient, with small key-sizes. Thus, this primitive has been widely used to construct cryptographic protocols. Post-quantum migration is largely concerned with replacing NIKEs with KEMs and Signatures. The famous Diffie–Hellman [44] and Elliptic Curve Diffie–Hellman [32] operations are NIKEs.

Signatures Post-Quantum variants: ✓ Signatures can be used to show that someone has placed their stamp of approval upon a particular piece of data. This can be used as a sort of digital passport or watermark.

KEM "Key Encapsulation Mechanism" Post-Quantum variants: ✓ This can be used to transfer a symmetric key to another party. Since post-quantum KEMs exist, a combination of KEMs and, in some cases, Signatures are used to replace NIKEs when a protocol must be upgraded to post-quantum security.

AKEM "Authenticated Key Encapsulation Mechanism" Post-Quantum variants: ✓ (Introduced in this work) This interface was introduced in the HPKE standard. It has capabilities in between those of a KEM and those of a NIKE. As with a KEM, a key not directly derived from two key-pairs is transferred, so interaction is required. Like a NIKE, AKEMs authenticate the sender of a key, which is missing from KEMs alone.

2.9 Comparing avionics and cryptography

Although the ultimate aims of cryptography and avionics remain similar, the fundamental approaches, taken by both fields, differ greatly. Whereas cryptography concerns itself with security properties, avionics is driven by the pursuit of safety. That divergence has created a parallel series of novel concepts that, nonetheless, often remain comparable.

One such divergence is cryptography's willingness to sacrifice availability of a message, if necessary to preserve confidentiality, integrity, and authenticity. On a public WiFi, it is not possible to ensure perfect availability, yet cryptography has well protected integrity, authenticity, and confidentiality in practice. After all, it is significantly less bothersome to be forced to reload a webpage than to have one's bank details leaked to all who may be interested.

In avionics, however, availability is everything. A loss of connection at a crucial moment can represent the start of an emergency situation, and safety regulations require critical control law[5] to be available at all times. Integrity must also not be compromised, given the critical nature of communications. Authenticity and confidentiality, however, are not prioritised in quite the same way. Often, the physical separation of electrical connections serve as the only major

[5] e.g. algorithms that convert side-stick movement to control surface deflections, ultimately steering the aircraft.

authenticity guard, and most aviation radio frequency protocols currently used do not feature effective measures to implement confidentiality or authenticity.[6]

One tried-and-true method for ensuring availability is redundancy: having multiple instances of a component, combined with a fail-over, increases availability in the event of faults. Unfortunately, systemic faults, like software-bugs, still retain the potential to affect availability, despite the use of redundancy. To address this risk, dissimilar hardware and software is used; by combining two separate implementations, systemic failure inherent to one implementation can be more easily caught.

Redundancy is a concept that is also favoured in cryptography: crypto-combiners that allow one to combine two different cryptographic schemes for improved security. This results in a hybrid secure system whereby, even if one of the schemes fails its security promise, the other scheme maintains security.

Both fields put exceptional effort into verification and validation, through certification, extensive testing, and formal methods. While a strong emphasis on rigorous testing is palpable in both fields, the priorities remain different. Sound cryptoanalysis is the basic marker of decent work in cryptography, and certification through a certification body is the absolute must-have in avionics.

In furtherance to this demonstration of similarities, non-functional behaviours, such as timing and memory usage, also receive special attention in both fields. Deterministic timing behaviour is a must to build robust real-time systems in avionics. If an algorithm suddenly takes significantly longer with a given input, safety properties will be at risk. A related scenario threatens security in cryptography: timing-based side channels where differences in an algorithms execution time leak secret information have been shown to cause many security vulnerabilities. To ameliorate the risk of surprises at runtime, and to keep secrets confidential, a deep understanding of the underlying hardware, along with ongoing and rigorous testing, is required.

The skillset required for successful avionics software engineering overlaps with that required for successful cryptography. In short, both require the ability to write software with a high degree of assurance that said software will meet the desired properties. The differences are found foremost in the means of reaching the acceptance of an implementation, with avionics relying on certification, and cryptography focusing on mathematical proofs.

[6] despite the RF spectrum being a broadly shared medium.

3 Related works

Previously in Sect. 2.9, we mentioned that the avionics sector does not prioritise confidentiality. Despite this, there have been multiple attempts to provide confidentiality, and other security oriented properties, for communication in avionics. This section introduces some of those existing approaches, their benefits, and their shortcomings.

3.1 ACARS

The majority of text-based communication to and from aircraft is transmitted using ACARS. First deployed in 1978, the protocol became the de-facto standard for short text messages between aircraft and the ground. Bandwidth for ACARS is limited, reaching up to 30 kbit/s for air to ground traffic, and up to 400 kbit/s on satellite backed links. Messages are rather small, with 228 B for uplink and 238 B for downlink messages. Both downlink and uplink messages allow for a 210 character long text which, however, is limited to the Baudot character set. Two primary users of ACARS are Air Traffic Control (ATC), to issue route clearances and airlines, and for fleet management activities such as distribution of flight plans. [1]

No cryptography is mandated or included in the original standard [1]. That is despite its utilisation for safety critical information (from ATC) and privacy sensitive (airline reporting and maintenance) information. ARINC 823 specifies an encryption layer, ACARS Message Security (AMS) [45]. While performing a cryptographic analysis of AMS, Blanchet found problems with it, which were communicated back to the industry editor of ARINC 823 [45]. Still, since the publication of [45] in august 2017, no new revision of ARINC 823 has been released. Furthermore, AMS is seldom used, in part due to cost, as AMS is charged extra on top of ACARS service fees [1]. Instead, many operators use a proprietary cipher for ACARS which relies on a monoalphabetic substitution cipher, which can only be described as security theatre [1].

In summary, ACARS itself is not secure, AMS which aims to address security in ACARS is not widely adopted, and the alternative in wider use today only creates a false sense of security. Performance wise, ACARS leaves much to be desired, such as longer messages comprising a broader selection of symbols.

3.2 LDACS

L-band Digital Aeronautical Communications System (LDACS) is an air to ground communication link aiming to address future communication needs in aviation. Its design is similar to that of Long Term Evolution (LTE), featuring

a cell oriented architecture: multiple ground stations each host a LDACS cell, and aircraft associate with a near-by cell. One cell can host up to 512 aircraft, over a range of approximately 100 km [46–48].

One goal of LDACS is the establishment of security at the link layer [48], and Mäurer, Gräupl, Schmitt, Rodosek, and Reiser clearly identified the threat of quantum computers to classic asymmetric cryptography, therefore, adding provisions for pre- and post-quantum cryptography in the LDACS cell-attachment protocol described in [46]. Due to concerns regarding the communication overhead, SIKE was elected as the post-quantum secure Key Encapsulation Mechanism (KEM), as it features comparatively small cipher text (*ct*) and public key (*pk*) sizes (see Table 1) [46]. Unfortunately, Castryck and Decru published a critical vulnerability in SIKE [49] approximately one year after the publication of the LDACS Cell Attachment paper [46]. The attack, an effective key recovery algorithm, renders SIKE unsuitable for any security related application. This demonstrates how important crypto-agility is; even when planing for future threats by utilisation of post-quantum security, algorithms can become insecure solely by external discovery.

Analysis of the proposed protocol, in combination with the specific selection of available cipher suites, allowed the author of [46] to asses and prove the feasibility of LDACS given the constraints of the physical link (such as bandwidth) as well as defining upper limits on timing.

As such, LDACS is the most promising contender for a future-proof radio-communication protocol in the avionics sector. While post-quantum security was aimed for, due to SIKE being broken, it was not achieved. Being air-to-ground based, LDACS is well equipped to take over use-cases from ACARS, but the lack of air-to-air communication leaves use-cases like ADS-B unresolved.

3.3 AeroMACS

Similar to LDACS, Aeronautical Mobile Airport Communication System (AeroMACS) aims to advance the state of air to ground communication [50, 51]. The predominant use-cases mentioned for AeroMACS are air traffic control and airline operations communications, like ACARS. To implement secure communication, a classical Public Key Infrastructure (PKI) relying on X.509 certificates is foreseen [46, 52]. As Mäurer, Gräupl, Schmitt, Rodosek, and Reiser point out, AeroMACS "only supports one cipher suite option" [46], which is therefore vulnerable to quantum computer attacks. We could not find any publication indicating a consideration of adding post-quantum cryptography to AeroMACS.

A PKI approach is sensible, to a point where Mäurer, Gräupl, Schmitt, Rodosek, and Reiser even propose a joint PKI for LDACS and AeroMACS [46]. X.509 certificates are compatible with post-quantum cryptography [53]. Apart from the PKI infrastructure described in AeroMACS, it, however, seems unsuitable for future use due to its current lack of post-quantum security and no roadmap leading towards it.

Table 1 Size in bytes of *pk*, secret key (*sk*) and *ct* for various cipher suites

	pk	*sk*	*ct*
SIKEp434	330	44	346
SIKEp751	564	80	596
ML-KEM512	800	1632	768
ML-KEM512-X25519	832	1664	800

4 Our contribution

First, we enhance Hybrid Public Key Encryption (HPKE) [19] with a post-quantum secure Authenticated Key Encapsulation Mechanism (AKEM) for hybrid security. Secondly, the now post-quantum-secure HPKE variant is integrated into an ARINC 653 partition. This integration utilises a Remote Procedure Call (RPC) API to enable any partition to use the post-quantum-secure HPKE without linking, nor directly depending upon, the cryptographic code. A small benchmark evaluates the impact on memory consumption and execution time, two key properties for integration into dependable real-time systems. At this point, we perform a demonstration of crypto-agility, by swapping the cryptography in the demonstrator, without touching the application partitions that use the cryptography at all.

4.1 Post-quantum security for HPKE

We construct two new AKEMs, that can be used in the context of HPKE, to provide post-quantum security. This is a hybrid construction, i.e. the construction redundantly uses a well-established pre-quantum cipher, together with post-quantum secure cryptography in order to remain secure even if one component fails.

The construction follows the recipes for encapsulation and decapsulation listed in Table 3; a detailed description of the encapsulation and decapsulation recipes can be found in figures Figs. 19 and 20. Templates for constructing the key data structures in the scheme are given in Fig. 2. A specification of the schemes we introduced in Rust-like pseudocode can be found in Appendix 4 and Appendix 5 with primitives used being specified in Appendix 3.

HPKE [19] is a fairly recent standard for asymmetric encryption that provides both authenticity and secrecy for

asynchronous one way communication (Sects. 2.6, 2.8). The standard uses KEMs "Key Encapsulation Mechanisms" and AKEMs—"Authenticated KEM", a type of KEM where the sender needs to authorise themselves in order to be allowed to transmit any data.

Both constructions given in this paper use the SHA-3 [54] variant SHAKE256 as a key derivation function, and they both start from X25519HkdfSha256. This is a cipher developed as part of HPKE that provides pre-quantum encryption and sender authentication. Combining a pre-quantum cipher with a post-quantum cipher in this manner serves to hedge our bets: Post-quantum cryptography has been standardised very recently and, while unlikely, there may be critical flaws in its design. If this proves the case, our cipher is still as secure as using X25519HkdfSha256 on its own. *No security is lost, only gained.*

We do not provide any proofs of security at this time, however we do provide a security argument in Sect. 6.

To build X25519MLKEM768, we add ML-KEM [42], a key-derivation-function [55] ("KDF") based combiner, instantiated with shake256 [54] as a key derivation function. That is, we use both key encapsulation mechanisms to transmit the key and then pass both keys to a KDF to derive a combined key. For technical reasons, we also include the X25519HkdfSha256 ciphertext in the key derivation step[7] and as a measure of heuristic security, we also include the X25519HkdfSha256 public keys.

This cipher provides sender authentication using pre-quantum cryptography, but it also provides post-quantum secrecy. It strikes a balance between efficiency and post-quantum security: the message can not be decrypted after transmission even if an attacker gained access to a quantum computer, but an adversary could impersonate the sender if they had access to a quantum computer right now. See Appendix 4 for a detailed description of the X25519MLKEM768 construction.

To also provide post-quantum authenticity—prevent an attacker with a quantum computer from impersonating the sender—we provide a separate variant that makes use of the *ML-DSA3* [43] signature scheme. The result achieves both post-quantum secrecy and post-quantum sender authentication. It retains all major security properties, even if the classical cipher used is completely broken.

[7] X25519HkdfSha256 does not provide ciphertext collision resistance [56]; i.e. it is possible for an attacker who knows the recipient secret key to generate two ciphertexts that decrypt to the same shared key. The mathematical models used to analyse the security of key encapsulation models contain a condition that leads to a theoretical attack under those circumstances [57]. It is likely that this quirk has no impact on the practical security of KEM combiners, but not accounting for this issue would force us to consider an alternative mathematical model of KEMs. Given the small impact of hashing 32 additional bytes of data, we prefer to stick to the established models.

This construction is not quite as straightforward as the previous one as giving an attacker access to a signature breaks anonymity [58]. To demonstrate this, consider this scenario: An attacker has a list of potential sender public keys. They intercept a message and would like to find out who the sender is. To achieve this, the attacker can simply try to validate the signature under each available public key. One of those keys will mark the signature as valid, giving away the identity of the sender.

To provide some measure of anonymity despite using a signature scheme, the signature is encrypted: We utilise further output from the Shake256 key derivation function as a stream cipher. For technical reasons (see Sect. 6) we also add a second stage of key derivation so we can also include the signature in the output key. This ensures that an attacker cannot figure out the sender identity, even if they know a list of sender public key candidates. An additional requirement is that the KEMs used (i. e. X25519HkdfSha256 and ML-KEM768) provide anonymity and secrecy; if one component is giving away the sender identity, hiding a different component cannot solve this. See Appendix 5 for the full definition of the X25519MLKEM768MLDSA construction.

The variant X25519MLKEM768 should be used in scenarios where data is exchanged now but has to remain secret for many years, as X25519 provides authentication in the present and ML-DSA ensures the secrecy of data even in the presence of an quantum computer. The variant X25519MLKEM768MLDSA should be used once cryptographicaly relevant quantum computers exist, as X25519 can no longer be used to ensure authenticity of the data and, therefore, ML-DSA is needed to ensure the authenticity of the data in the presence of a quantum computer.

Our construction is generic; i.e. it can be used to combine any number of KEMs, AKEMs, and signatures using a key derivation function. We note though, that researchers building other combiners based on our construction need to be careful to check which of the constituent ciphers provide ciphertext collision resistance [56] and which of the signatures provide uniqueness [59].

We conjecture—perform a well-reasoned mathematical guess—that this construction provides secrecy as long as at least one of the KEMs or AKEMs provide secrecy. The combiner is hypothesised to provide authenticity as long as the combiner provides secrecy and at least one of the AKEMs or Signatures provides authenticity (see Sect. 6).

The various ingredients used in the construction of our post-quantum secure encryption scheme. Our solution is specific to HPKE, the standard we extend, in that all other ingredients could be replaced with alternatives, providing either more security or more performance.

HPKE [19] The encryption standard we extend The HPKE standard still handles most aspects of encryption scheme

construction. We merely build new post-quantum secure key-transferral techniques (Sect. 2.8) for use within HPKE. See Sects. 2.6 and 2.8.

X25519HkdfSha256 [19] The pre-quantum KEM All constructions studied in the work combine two AKEMs (Sect. 2.8): one that is pre-quantum secure, and one that is post-quantum secure. X25519HkdfSha256 is part of the HPKE standard; it is based on very well-studied cryptography and provides a baseline of security, in case our post-quantum secure AKEMs fail.

SHAKE256 [54] The Key Derivation Function As we combine multiple key-transferral schemes, we thus must combine the resulting keys into a single key. The key derivation function takes care of that. We use the first output from the SHAKE256 function as our output key. In the variant with post-quantum authenticity, we use further output from the KDF to encrypt the signature, hiding it from observers to enable some measure of anonymity. SHAKE256 is part of the standardized SHA-3 has function.

ML-KEM768 [42] The post-quantum KEM ML-KEM is the standardized post-quantum secure key transferral mechanism. It comes in three variants, and we chose the middle variant as a balance between speed and security. ML-KEM is what endows our constructions with post-quantum secrecy.

ML-DSA3 [43] The post-quantum signature ML-DSA is one of the standardised post-quantum signatures. ML-DSA3 is the middle variant again: not the fastest but not the highest security margin either. The post-quantum signature is used in our most advanced solution, providing all the security that the pre-quantum variant provides. It ensures that the sender of a message cannot be impersonated.

The various ingredients used in the construction of our post-quantum secure encryption scheme. Our solution is specific to HPKE, the standard we extend, in that all other ingredients could be replaced with alternatives, providing either more security or more performance.

4.2 Integrating HPKE in an ARINC 653 partition

As outlined in Sect. 4, one of our primary goals is to enhance crypto-agility. To achieve this, a modular approach is beneficial; if all the cryptography is a black box module with a defined interface, replacing it becomes simple. Thus, and in accordance with IMA design paradigms, we implemented said module as a generic crypto-partition. Available over sampling ports, this partition can not only both encrypt and sign (`seal`), but also decrypt and verify (`open`) messages for other partitions. The crypto-partition can serve multiple other partitions, so it is sufficient to embed only one crypto-partition per hypervisor.

The interface available to other partitions does not reveal the particular KEM in use. A swap of the crypto-partition is, however, implicitly observable to normal partitions over a change of the *ct* size for a given plain text *pt* or just a differing *pk* size. The crypto-partition's interface (see LST 1) comprises the standard operations of HPKE but handling of the *sk*—*sk*s never leave the crypto partition. This keeps the scope of high assurance code reasonably small; only the crypto-partition is responsible for keeping *sk*s secure.

LST 1. Minimum viable selection of cryptographic operations. "Open" verifies a message authenticity, thus it is possible that either one or no *pt* is returned

```
setup() -> pk
seal(pk_peer, pt) -> ct
open(pk_peer, ct) -> pt?
```

When compared to simply linking cryptography code wherever needed, a crypto-partition has a number of benefits. Keeping track of the cryptography code is simple because it resides in one place. Maintenance tasks, such as updating the cryptographic primitives, are simplified by the fact that only the crypto partition is touched. Certification of the crypto-partition can benefit from approval as RSC: once certified for one aircraft type, a crypto-partition can be embedded into another aircraft type with minimal effort on module acceptance. As its functionality is highly generic and not connected to a specific aircraft functionality, we anticipate that most of the previous certification evidence can be re-used. When changing the crypto-partition (for example to replace a vulnerable cipher suite), the module/application acceptance of other partitions that use the crypto-partition remain untouched. That is, the code of other partitions does not change, thus evidence regarding objectives related to that code stays valid. Proper composition (system acceptance and aircraft integration) do of course still need to be demonstrated after every change of *any* partition, as outlined in Sect. 2.2.

Our demonstrator comprises three partitions, which run in our a653rs-linux hypervisor. All three of them are implemented in Rust, and can be ported to any hypervisor with a653rs support with trivial effort.[8] Two normal partitions, `sender` and `receiver`, use HPKE operations offered by the third one, `crypto_part`. Multiple instances of the third one can be used, there is no reliance on using a shared crypto-partition instance. Messages between the partitions

[8] subject to the processor architectures supported by liboqs [61], in particular x86, aarch64, armv7, ppc64 & s390x.

are exchanged via sampling ports, a directed flavour of shared memory between partitions.

Figure 3 depicts an exemplary exchange of a secret message between the sender and the receiver partition. The sender first requests a `seal` operation from a crypto-partition, which transforms the *pt* into a *ct*. This *ct* is then sent to the receiver partition which, using the crypto-partition's `open`, decrypts the *ct*. During decryption, the crypto-partition ensures that the message is indeed from the sender partition, so `open` assures authenticity. Using this setup, we swapped in different crypto-partition implementations with a variety of alternatives and the original form of HPKE. In particular, a fork of the Rust hpke crate[9] was extended with implementations of cryptographic algorithms from liboqs[10] [61]. In all cases, the message could be sent from the sender to the receiver partition without issues. As planned, no change in the sender and the receiver partition's code was necessary—the specific cryptography provided by the crypto-partition is opaque to the other partitions.

One notable gap remains: in this form, we assume that both sender and receiver already know each other's *pk*. This is a non trivial requirement, and fulfilling this for example through a PKI is a significant amount of work that escapes the scope of this paper.

5 Evaluation: performance and memory overhead

To evaluate our proposed changes on HPKE, we measured both the memory and time overhead induced by our post-quantum secure HPKE variants. Both memory and computational performance are important in an avionics context, as partitions are allocated fixed amounts of computation time and memory, which cannot be exceeded. Our measurements were conducted on an Intel i9-13900K processor, running Linux 6.6.54, without an ARINC 653 execution environment. We decided to forgo the use of an ARINC 653 hypervisor while benchmarking, as we found the hypervisor itself induced only a negligible overhead. The use of fixed memory and cpu time allocations also hinders performance measurements. To determine the CPU performance, citerion.rs[11] was used. The memory consumption was measured using the `softlimit` utility from the daemontools[12] software package. Using `softlimit`'s `-a` and `-s` options, the minimum acceptable overall memory and stack size for each variant of HPKE to `seal`/`open` a message was determined. All memory usage measurements were conducted with statically compiled minimal binaries for each variant of HPKE and each operation (`seal`/`open`). We recognize that these measurements heavily depend on the particular hardware and software environment provided by the test setup, particularly because the implementations tested here use CPU-specific optimization methods. In addition, we did not conduct a thorough survey of different implementations of any of the algorithms, so the performance achievable by using the fastest available software library remains unknown. Since the standardization of ML-KEM and ML-DSA are recent events, additional implementations of the schemes are being added at a rapid pace. To allow replication of our benchmarks in different measurement environments with potentially updated cryptographic implementations, we are making our measurement setup available as part of the supplemental material (see Sect. 8). Despite these limitations, the setup still provides a decent estimation of whether post-quantum adaption is infeasible as a result of runtime overhead.

The CPU performance measurements (Fig. 4) suggest that adding ML-KEM for post-quantum secrecy introduces a minor performance overhead for both seal and open operations (`seal`: 11 %, `open`: 40 %). Providing post-quantum authenticity by adding ML-DSA(+ML-KEM) comes at a greater performance overhead for either (`seal`: 90 %, `open`: 91 %). We should point out that these results are quite good: remember that we are using pre-quantum encryption operations in concert with post quantum ones; this analysis indicates that most of the CPU-overhead is encountered in the pre-quantum portions of our schemes. Raw data is available in the supplemental material (Sect. 8).

The liboqs [61] implementations of ML-KEM and ML-DSA seem to be optimized for CPU performance at the expense of memory efficiency because the memory overhead induced by our schemes (Figs. 5, 7) is much more severe in the seal but not the open operation. Using ML-KEM to provide post-quantum secrecy requires five times more memory in the `seal` operation compared to the pre-quantum secure variant while the `open` operation comes in at a modest 16% overhead (`open`: 544% or 4436 kB extra overhead; `seal`: 16% or approximately 128 kB). Using ML-DSA in addition to ML-KEM for post-quantum authenticity bumps the peak memory consumption by an almost negligible amount (`seal`: 3% or 156 kB; `open`:8% or 76 kB). All post-quantum operations show a increased use of stack memory (Figs. 6, 8) although the increase in stack-size is minor compared to the total increase in memory requirement. The biggest increase is seen in the variant of `seal` with both ML-KEM and ML-DSA, which needs roughly four times more memory than the variant without post-quantum security (pre-quantum: 44 kB; post-quantum: 212 kB). Raw data is available in the supplemental material (Sect. 8).

[9] https://crates.io/hpke.

[10] https://openquantumsafe.org/liboqs/.

[11] https://github.com/bheisler/criterion.rs.

[12] http://cr.yp.to/daemontools.html.

For small embedded systems, which only provide a couple hundred kB of RAM, the additional memory requirements for using post-quantum HPKE may impede the adoption of the post-quantum secure HPKE variant. However, computers in IMA have been equipped with at least dozens of MB of RAM since the early 2010s. We therefore conclude that utilization of our fully post-quantum secure HPKE variant in current avionics hardware is neither prohibited by either the runtime (Fig. 4) nor the memory (Fig. 5) overhead.

A prior version of our paper featured variants with much lighter memory requirements (see); we chose to migrate towards liboqs variants despite this, because the library enjoys wide usage and implements the ML-KEM and ML-DSA standards. Our results might indicate, though, that there is a need for greater focus on memory efficiency in the optimization of post-quantum cryptography.

6 Evaluation: security

While we do not provide a formal proof of security for our scheme, we did conduct a security analysis. In this section, we will outline the security argument of our scheme; that is, we outline the starting point for creating a formal proof.

First, let us revisit the basic types of attacker we must consider in our security argument: The passive, eavesdropping attacker and the active attacker who can change the network transcript of our cryptographic protocol. We generally assume attackers are active in the rest of this analysis.

The basic types of attackers used in the analysis of cryptographic schemes. We consider active attackers.

Passive An attacker who can observe all messages being transmitted on the network.

Active An attacker who can observe, change, drop, retransmit, and insert entirely new messages. In particular, an active attacker can perform a man in the middle attack.

Recall that we are constructing two authenticated KEMs with various levels of post quantum security. The basic security properties provided by a KEM are secrecy and authenticity.

Our security properties.

Secrecy An attacker should be unable to learn any information about the secret being transmitted.

Authenticity An attacker should be unable to perform a key exchange with another party and pass off its secret as somebody else's. In particular, active attackers should be unable to perform a man in the middle attack.

Both X25519MLKEM768 and X25519MLKEM768MLDSA are hybrid constructions: One is made up of a pre-quantum authenticated KEM and a post-quantum unauthenticated KEM, the other also includes a post-quantum signature scheme. Since we wish to show that our schemes provide redundant security, i.e. retain some security even when some of its components are considered insecure. To this end we divide our constituent schemes into two groups: Post-quantum schemes (ML-KEM and ML-DSA) and pre-quantum schemes (X25519HkdfSha256). To show security in the pre-quantum setting we can not rely on the security of ML-KEM and ML-DSA; to show our scheme is secure in the post-quantum setting, we cannot rely on X25519HkdfSha256.

The basic scenarios we must consider when analysing our scheme.

X25519MLKEM768, pre-quantum The scheme should provide all security properties (secrecy, authenticity, and identity hiding) against active adversaries without relying on ML-KEM for security.

X25519MLKEM768, post-quantum The scheme should provide secrecy and identity hiding, but not authenticity against active adversaries without relying on X25519HkdfSha256.

X25519MLKEM768MLDSA, pre-quantum The scheme should provide all security properties without relying on ML-KEM for security.

X25519MLKEM768, post-quantum The scheme should provide all security properties against active adversaries without relying on X25519HkdfSha256.

Note that both our authenticated KEMs can be used in an unauthenticated mode by not specifying any sender keys. In this mode, neither scheme provides authenticity.

We start by analysing X25519MLKEM768 relative to the security of X25519HkdfSha256. Since in this case, X25519HkdfSha256 is a secure authenticated KEM, the adversary is not able to obtain the shared secret generated by X25519HkdfSha256. As this shared secret is used as an input to the key derivation function, generating the final shared secret, the attacker is not able to compute the final shared secret, so the scheme provides secrecy.

To break authenticity, the adversary would have to either generate its own X25519HkdfSha256 keypair and generate a ciphertext under that key or use reuse the X25519HkdfSha256 ciphertext from a third party.[13] Then, the adversary

[13] Possibly gained by eavesdropping on some legitimate key transferal session.

would have to tell the recipient to decapsulate the ciphertext using another sender key. By the authenticity property of X25519HkdfSha256, this decapsulation step would fail, detecting the substitution. Therefore, the authenticity of X25519HkdfSha256 confers authenticity to our composite scheme.

The security argument for secrecy in the post-quantum is similar to the one in the pre-quantum case. Since we assume the secrecy of ML-KEM, the adversary is not able to obtain the shared secret generated by ML-KEM and, therefore, cannot compute the shared secret of the composite AKEM. In the post-quantum case, X25519MLKEM768 does not provide authenticity.

The security arguments for X25519KEM768MLDSA are similar to the arguments for X25519MLKEM768. Firstly, we have to take the two-stage key derivation process into account, but this does not change any of the previous arguments. Secondly, this scheme also provides post-quantum authenticity by using a ML-DSA signature, used to sign a key commitment. To break authenticity, the attacker would again need to produce a ciphertext of its own (or get a third party to produce a ciphertext). The attacker can successfully generate a X25519HkdfSha256 ciphertext, because in the pre-quantum scenario, we assume the scheme to be insecure. The attacker also succeeds at producing a ML-KEM768 ciphertext because it just needs the recipients public key for that purpose. The adversary can then produce a valid shared key and a valid key commitment, but it can not produce a ML-DSA signature using the identity it wishes to impersonate. Since no valid signature could be produced, the recipient detects this during decapsulation and aborts. Thus, the security of the ML-DSA signature scheme confers authenticity to our composite scheme.

Finally, there is a last attack scenario that we have to take into account to show that our scheme is secure. Giacon, Heuer, and Poettering[57] figured out that the IND-CCA [14] security cannot be achieved without mixing the components' ciphertexts into the key derivation step. The reasons for this are subtle, highly technical and possibly irrelevant for real world attack scenarios, but cryptographic analysis should generally stick to established security notions, so this attack is nonetheless relevant. This was refined during the construction of the X-Wing KEM [56] with hybrid security where it was established that this rule can be sidestepped by using KEMs with Ciphertext Collision Resistance: The attack introduced by Giacon, Heuer, and Poettering relies on finding some other ciphertext that decapsulates to the same shared key; i.e. it works by showing that the ciphertext is malleable (can be modified by the attacker). This is prohibited, because it was shown that malleable encryption schemes cannot achieve the highest level of IND-CCA security [62].

When a KEM ciphertext is just composed of multiple KEM ciphertexts, each of the subschemes providing IND-CCA security themselves, finding a collision should be impossible. Remember: The constituent schemes provide IND-CCA security, so they are not malleable, so the composite scheme should not be malleable either.

Except that when building robust combiners [18], we always assume some of our schemes are insecure. The attacker can gain access to their secret keys. Some of the schemes do not provide IND-CCA security so the mathematical imperative that previously led us to believe that a collision cannot be found, is gone. Ciphertext collision resistance is about regaining that imperative, even when IND-CCA security is gone and when the secret keys of a scheme are available to the attacker. Signature uniqueness is similar to ciphertext collision resistance, but the property applies to signature schemes instead of KEMs.

Our task now is to show that for both our combined AKEMs, changing one ciphertext leads to a different key, even if that ciphertext belongs to an assumed broken scheme. Recall that the X25519MLKEM768 ciphertext has two fields: The ciphertext belonging to X25519HkdfSha256 and the one belonging to ML-KEM. The X25519MLKEM768MLDSA ciphertext has one additional field, the encrypted ML-DSA signature.

X25519HkdfSha256 does not provide ciphertext collision resistance, but we mix its ciphertext into the first key derivation step. By the collision resistance property of our hash function, this field is covered.

ML-KEM provides ciphertext collision resistance, according to the X-Wing analysis [56], so this field cannot lead to a collision.

The ML-DSA signature is slightly more complex. ML-DSA does not provide uniqueness [59], but we encrypt the signature under key derived from both X25519HkdfSha256 and ML-KEM ciphertexts (albeit without authentication), so both in the pre-quantum scenario and the post-quantum scenario, the signature is encrypted. To cause a collision on ML-DSA, the attacker would need another, different signature for the same key commitment. To replace the encrypted signature, the attacker would have to compute the original signature too and apply an exclusive-or operation to the original encrypted signature. This operation would produce the keystream that was initially used by the sender to encrypt the signature, so now the attacker could use the keystream to encrypt its replacement signature.

This attack is not possible, because the adversary lacks access to the intermediate key, derived during the first round of key derivation, so the attacker cannot sign it. Despite this attack vector being closed, we still cannot exclude odd

[14] This is the mathematical model used for the security of key encapsulation mechanisms. It spells: Indistinguishability under Chosen Ciphertext Attack.

attacks such as an adversary's ability to derive a pattern of bit-flips that will not prevent the signature from being verified with some non-negligible probability.[15] This attack might seem contrived, but the security notion excluding it does not exist, so we cannot use this assumption.

There are two ways to fix that issue: We could use authenticated encryption, thereby rendering it impossible for the adversary to modify the signature without knowing the key, or we could use a second round of key derivation and mix the signature itself into the final key.

We opt for the second option since this also imbues our own combined scheme with ciphertext collision resistance: Two of our cipertext fields are hashed into the output key, the third is protected by ML-KEM's ciphertext collision resistance.

With this analysis, we are confident that the proposed AKEM combiner delivers on its security claims.

7 Conclusion

In this paper, we have demonstrated the integration of a state-of-the-art cryptography solution in an avionics environment, as well as the extension of this solution to provide post-quantum security. To this end, we first exposed an off-the-shelf Rust-programming-language software library that implements the HPKE (see Sect. 2.6) asymmetric encryption standard in an ARINC 653 partition (Sect. 4.2).

In order to ensure that the solution provides post-quantum security (see Sect. 2.7), we devised two quantum resistant authenticated key-transferral (see Sect. 2.8) schemes (Sect. 4.1). One scheme uses the standardised, quantum-resistant key-transferral scheme ML-KEM to post-quantum secrecy but not sender authentication. The other scheme combines ML-KEM with the standardised post-quantum signature scheme ML-DSA to also provide post-quantum authenticity. All of our quantum-resistant ciphers integrate a pre-quantum cipher to provide for cryptographic redundancy.

Performance measurements (Sect. 5) provide initial evidence that the memory-requirements and performance characteristics may be suitable for deployment in the flight-computers already being used. Even though, our strongest, post-quantum secure cipher, peak RAM usage was fairly high with about 5.5 MB and a stack usage of approximately 212 kB, it is still acceptable for current avionics hardware. Performance measurements are hard to give in absolute numbers. Nonetheless, all operations finished in under a millisecond on a single core of a modern CPU, yielding no evidence of a prohibitive performance-issue.

As a control, the same performance measurements were applied to a pre-quantum cipher. Transitioning from pre-quantum secure ciphers to post-quantum alternatives increased memory usage (factor 3.5), stack size (factor 40) and runtime (factor 4.9). All factors given here refer to the worst offender metric.

No effort was spent on performance-optimisation. This leaves the significant potential to improve the efficiency of our construction, particularly in regard to stack size. A good place to begin would be deciding upon the most performant variants of the quantum-resistant KEMs and signatures.

8 Outlook

During our preliminary research, we were disappointed to find a meagre body of literature detailing the connections between avionics and cryptography. Indeed, the works we did find were predominantly cryptoanalysis papers with discouraging results, pinpointing severe flaws in the protocols used in the day-to-day operation of avionics systems. Thus, in the course of writing this paper, we necessarily became better acquainted with the areas, methods, and techniques in which avionics and cryptography meet.

Nonetheless, one paper, on improving the security of LDACS, does stand in contrast to those findings. It employed a great number of excellent techniques, even providing for a post-quantum secure solution. Unfortunately, along with stopping short of providing a fully-fledged proof of security, it employed a less-conservative, albeit highly efficient, cipher that had an attack published just one year after the paper's publication.

Although replacement of that insecure cipher is a technically easy task, this story does showcase the need for tighter integration between avionics and cryptographic communities. It also demonstrates the aspect of cryptographic work most unfamiliar to avionics specialists: the need to move quickly, reacting to new results in the field by deploying timely upgrades to meet new threats. In other words: crypto-agility.

Cryptographic agility requires adapting standardisation processes, certification procedures, software-upgrade infrastructures, and the technology itself to enable the most efficient possible turnaround time.

Making headway on that particular problem necessarily involves the evaluation of the gap between avionics execution environments and bleeding-edge cryptographic software. Standardisation bodies, engineers, and certification agencies need to know whether entirely new infrastructure must be built, or whether the avionic computers in use today are suitable for the task. Similarly, software

[15] "Negligible probability" is the mathematical jargon cryptographers use to describe events that are unlikely to the point of impossibility in relation to some security level.

developers within avionics must be able to estimate just how deeply avionics software should be integrated into their existing code bases, multiplying the efforts needed to upgrade and re-certify their code.

We began with the conviction that today's planes are capable of running state-of-the-art cryptographic software and also that post-quantum secure cryptographer should be employed from the start. As for achieving crypto-agility, we stand to benefit from the body of avionics research that has already gone into improving the separation between various software components in the name of improving turnaround times.

A demonstrator of this approach was devised: something small, with a minimal set of features, and broad applicability. Its cryptographic solution should be standardised, and it should provide for a simple interface to improve the component's usability. Serendipitously, the HPKE standard for one-way encryption and decryption has been recently published. It features two simple operations, seal and open, which can be used for secret, trusted message transmission in a broad range of applications.

To showcase the viability of this standard within avionics, we wrapped a standard implementation of this interface into a partition, following the standard isolation scheme used across avionics. Rather than devise a wholly new interface, we simply exposed the HPKE operations; keeping feature sets small to accelerate certification. This setup facilitates crypto-agility as the cipher can be exchanged by replacing the cryptographic partition.

Post-quantum security was regarded as a hard requirement for our demonstrator. Therefore, we proposed updated variants of the ciphers underlying HPKE, with the first employing state-of-the-art classical cryptography via the cipher developed as part of the HPKE standard itself. The other two variants then combine this standard cipher with post-quantum-secure ciphers to achieve either secrecy or both secrecy and authenticity.

All variants were deployed within the hypervisor, and exchanging those variants posed no particular challenge. To aid integrators in gauging the performance and memory requirements of these solutions, we collected a series of initial performance measurements. We found that, in comparison to the dummy-cipher, our most advanced solution requires below 5 MB of additional RAM, but does come with a large stack size requirement of below 300 KB. Reducing the stack size should a focus of future research.

Although we cannot provide absolute measurements for performance, as this will vary greatly between platforms, we found that on our modern hardware, all operations finish in less than a millisecond. Our most advanced post-quantum secure cipher was less than two times slower than the standard cipher it was compared against. There remains a lot of headroom for efficiency gains in our setup, too. For instance, other variants of our post-quantum ciphers boast a smaller security margin but a higher memory-efficiency and thus better performance. Avionic computers built in the 2000s possess only dozens of megabytes of RAM, and yet our solution certainly does not cross that boundary.

As usual, more research is required and, in the case of our demonstrator, we would recommend looking at its performance on computer models matching those used in real aeroplanes. We would also suggest a closer look at the problem of key distribution, for instance via the integration of a key signing infrastructure, such as the TLS certificates used broadly on consumer web browsers.

Table 2 Security properties of our constructions compared to the HPKE standard construction *X25519HkdfSha256* The first column shows the standard construction, providing no post-quantum security at all. The second column shows the standard construction plus ML-KEM, providing additional, redundant postquantum secrecy. The third column shows the HPKE standard construction plus both ML-KEM and ML-DSA, providing the full set of redundant post-quantum security properties: Secrecy and authenticity

	HPKE Standard	+ML-KEM	+ML-KEM +ML-DSA
Secrecy			
Classical	✓	✓	✓
Post-Quantum	×	✓	✓
Authenticity			
Classical	✓	✓	✓
Post-Quantum	×	×	✓

Appendix 1: Comparison to preprint benchmarks

The preprint [64] of this paper featured an implementation of our combiner using alternative primitives: Since the preprint was published before the ML-KEM and ML-DSA standards were released, we used ML-KEM and ML-DSA[16] variants from round 3 of the NIST post-quantum cryptography competition. Their implementations had a few advantages, including much reduced memory footprint. These implementations were also written in pure Rust, which simplifies deployment. We opted to migrate towards liboqs [61] based implementations of ML-KEM and ML-DSA despite their disadvantages because we preferred using the standardized variants and liboqs provides relatively well-trusted implementations. Comparative benchmarks are given in Figs. 9, 10, 11 and 12. The liboqs variant has better CPU performance, much worsened memory usage, though the use of stack memory improved slightly.

[16] Kyber was renamed to ML-KEM after standardization, Dilithium was renamed to ML-DSA.

Table 3 The steps to construct a new KEM or AKEM with redundancy using our recipe [60]

		Encapsulation	Decapsulation
1	KEM	For each KEM/AKEM, call encap() with one's corresponding own secret key and the peer's public key	For each KEM/AKEM, call decap() with one's corresponding own secret key, the peer's public key, and the corresponding ciphertext
2	Inner KDF	Input a unique domain separator [60] and all the decapsulated secrets into the KDF. Include the ciphertexts of any constituent KEMs that lack ciphertext collision resistance [56].	
3	Inner KDF (intermediate key)	Extract a 32 byte output key—the intermediate shared secret—from the key derivation function	
4	KDF (key commit.)	Extract a 32 byte key commitment from the key derivation function	
5	Signature decryption		XOR all signatures with further output from the KDF
6	Signature	For each signature scheme, generate a signature. If the AKEM is used in sender-unauthenticated mode, set all the signatures to zero	For each signature scheme, validate the signature. If the AKEM is used in sender-unauthenticated mode, ignore the signatures
7	Signature encryption	XOR all signatures with further output from the KDF	
8	Outer KDF	Input a unique domain separator [60] and the intermediate key. Include the signatures of any constituent signature schemes that lack uniqueness [59].	
9	Outer KDF (output key)	Extract a 32 byte output key—the shared secret—from the key derivation function	
10	Return	The output key, each KEM ciphertext, and each encrypted signature	The output key

If no signature scheme is used or if all the signature schemes used provide uniqueness [59], then the second key derivation step can be omitted, and the intermediate key can be used in place of the output key. If two key derivation steps are used, use dedicated domain separators to prevent oracle cloning attacks

These results lead us to conclude that there is much room for alternative optimization targets such as memory use in future implementations of ML-KEM and ML-DSA. The data from the preprint does not match the values presented in this comparison because the benchmarking setup has been changed since the benchmark was released. The values shown in the figures were all generated on the new benchmarking setup.

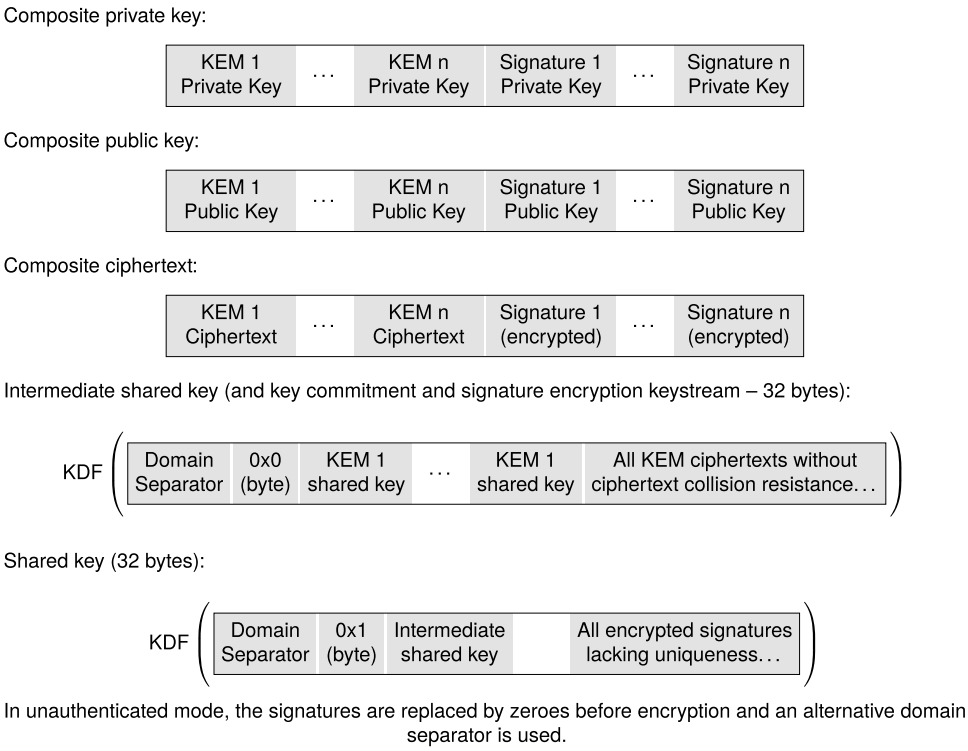

Fig. 2 The layout of structures used in our combiner

Appendix 2: Security category 5 variant

In this paper, we have been using the ML-KEM variants for what is called "security category 3"; this is an excellent choice for most applications. These security categories are defined by NIST [65]; category 1 is supposed to provide the equivalent security level of AES128 [66], with category 3 and 5 specified to provide the equivalent security of AES192 and AES256, respectively. ML-KEM [42] and ML-DSA [43] do provide support for security level 5 and so it is interesting to see how the security of our scheme would change if security category 5 primitives are used. To avoid relying on the collision-resistance of ML-KEM in our scheme, we use the combiner from Giacon, Heuer, and Poettering [57] instead of an adapted variant of the combiner introduced in X-Wing [56]. To achieve an improved level of security in the pre-quantum variant, we also use `DHKEM(X448, HKDF-SHA512)` [19] instead of `DHKEM(X25519, HKDF-SHA256)`; i.e. we use the higher-security curve 448 [67] curve instead of curve 25,519 in our Diffie-Hellman primitive. The results can be inspected in Figs. 13, 14, 15, 16, 17 and 18. Use of the GPH combiner generally has a very minor impact on all metrics. Use of the category 5 security variant has a large impact on the CPU performance of our cryptographic scheme, a very minor impact on memory usage and a moderate impact on minimum stack requirement. The decrease in CPU performance seems to be driven by the change in pre-quantum schemes (migration from x25519 to x448) whereas the increase in minimum stack requirement is driven by the post-quantum schemes (migration to category 5 variants of ML-KEM and ML-DSA).

Appendix 3: Primitives for protocol construction

AEAD

AEAD stands for "Authenticated Encryption with Additional Data". AEADs are the interface used for symmetric encryption; since no asymmetric cryptography is used most existing AEADs are post-quantum secure.

key	"k"—The cryptographic key
nonce	"n"—Nonce. A Number used once. This can either be a counter or a random number if the number is at least 192 bytes long.
payload	("pt" for "plain text" or "ct" for "cipher text")—The actual data being transmitted.
additional data	"ad"—Extra data that both participants should agree on, such as the name of the sender and the receiver so a mismatch can be detected.

```
AEAD.encrypt(k, n, pt, ad) -> ct
AEAD.decrypt(k, n, ct, ad) -> pt or null
Rule:
  AEAD.decrypt(k, n, ct, ad) =
    if AEAD.encrypt(k, n, pt, ad) = ct
      then return pt
      else return null
```

Null is returned if there is any mismatch in the `k`, `n`, `ct`, or `ad` parameters. That is, AEADs provide *authenticity* and *integrity*.

NIKE

NIKE stands for "Non Interactive Key Exchange". NIKEs provide the most powerful features for cryptographic protocol design, but attempts to provide a post-quantum secure variant have failed.

The challenge of migrating to post-quantum security is mostly about replacing NIKEs with KEMs.

```
NIKE.keygen() -> (sk, pk)
NIKE(sk, pk) -> k
Rule:
  NIKE.nike(sk1, pk2) = NIKE(sk2, pk1)
    if both (sk1, pk1) and (sk2, pk2) where each properly generated using NIKE.keygen().
```

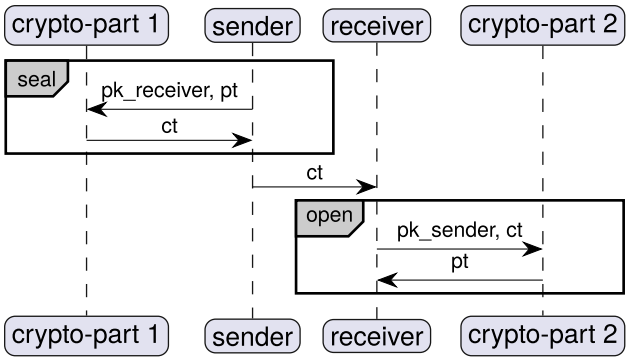

Fig. 3 Interaction between the three partition in the demonstrator

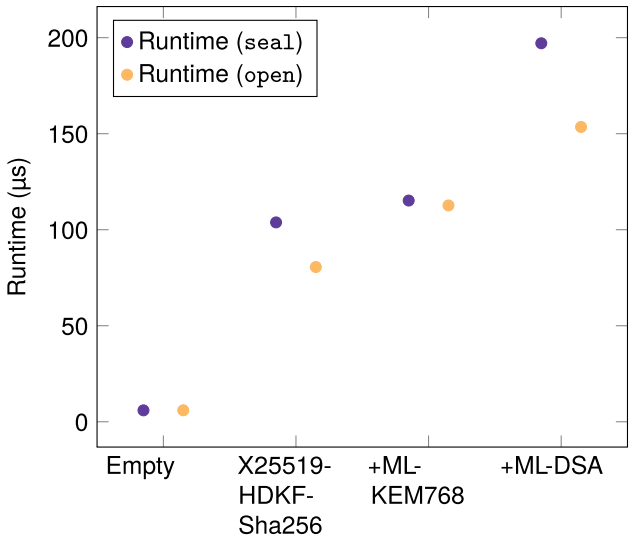

Fig. 4 Run-time of HPKE's main operations

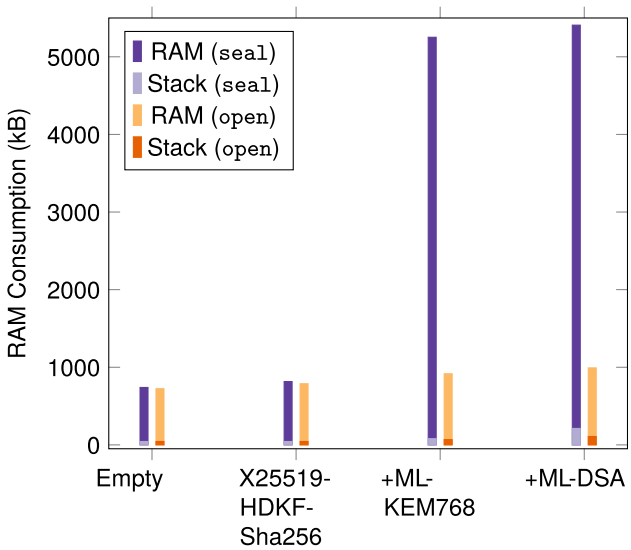

Fig. 5 Memory consumption of HPKE's main operations. RAM measures the total memory required (including machine code, static sections, etc.), while *Stack* only refers to the minimum allowable stack size for execution

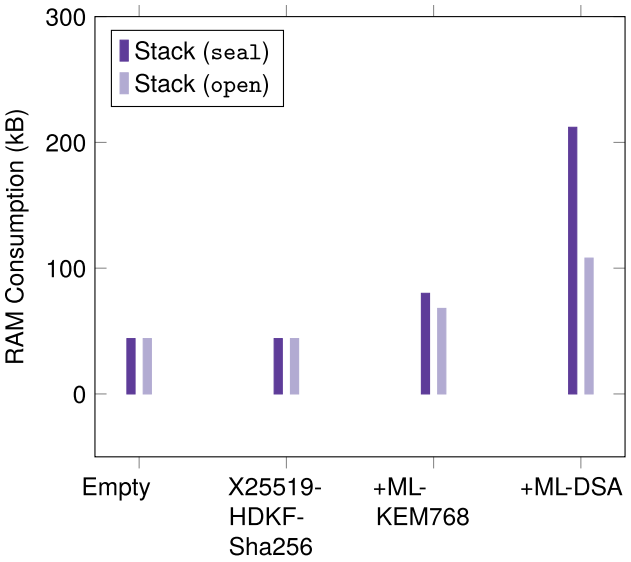

Fig. 6 Minimum allowable stack size of HPKE's main operations for execution

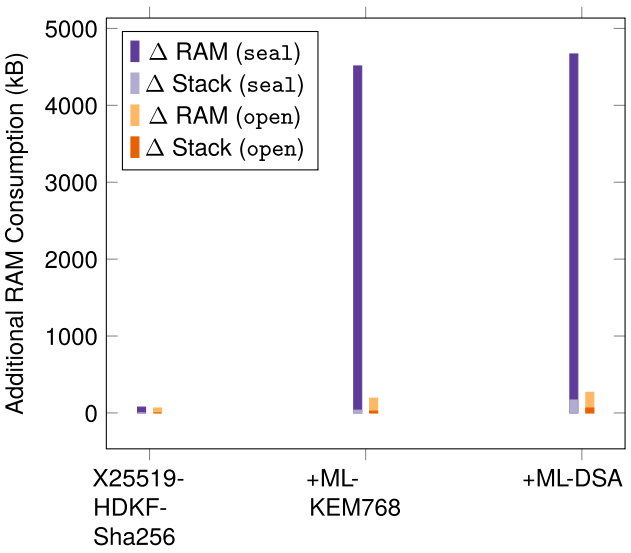

Fig. 7 Memory consumption of HPKE's main operations, relative to Empty. RAM measures the additional memory required (including machine code, static sections, etc.), while *Stack* only refers to the increase in minimum allowable stack size for execution both compared to the empty case from Fig. 5

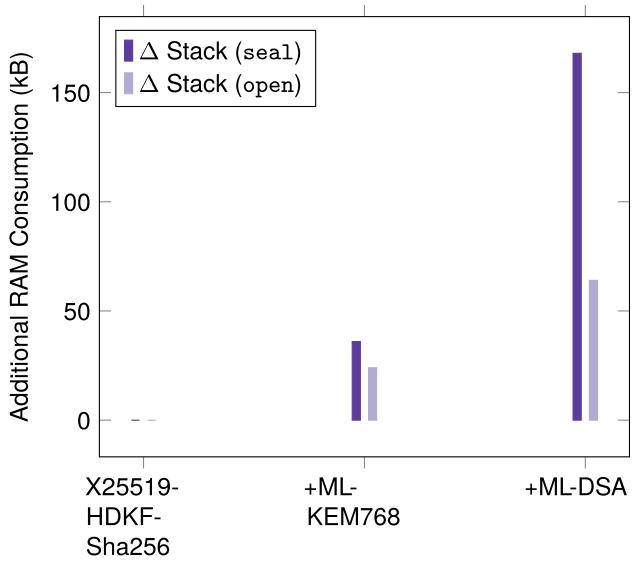

Fig. 8 Increase in minimum allowable stack size of HPKE's main operations, compared to the empty case from Fig. 6

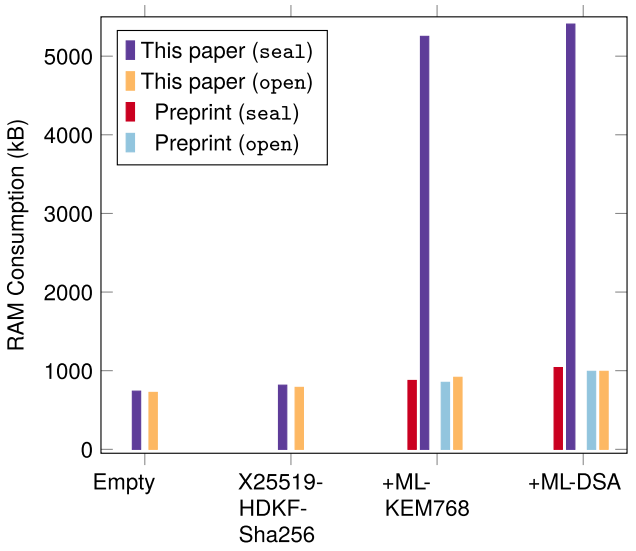

Fig. 10 Memory consumption of HPKE's main operations. Comparison between preprint version and version from this paper

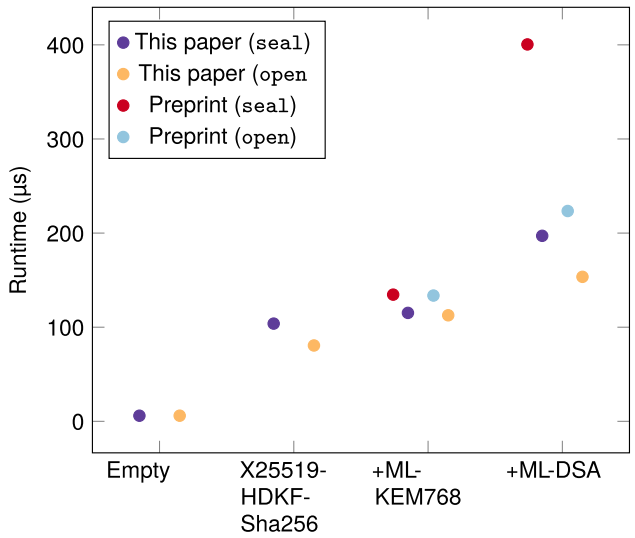

Fig. 9 Run-time of HPKE's main operations with comparison between the implementations from the benchmark and from this paper

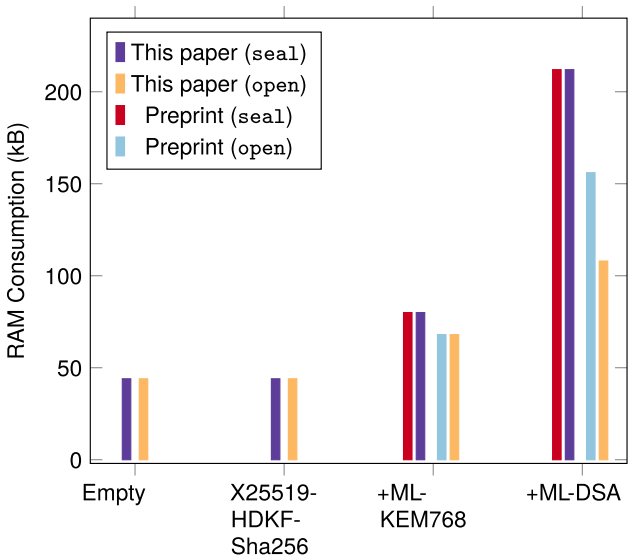

Fig. 11 Minimum stack requirement of HPKE's main operations. Comparison between preprint version and the version from this paper. Observe that the required stack size barely differs between the versions of our paper, with the exception of the open option in the ML-DSA variant, which improved by about 30 % absolute (or 57 % after subtracting the stack usage of the dummy—"Empty"—implementation) when migrating to liboqs

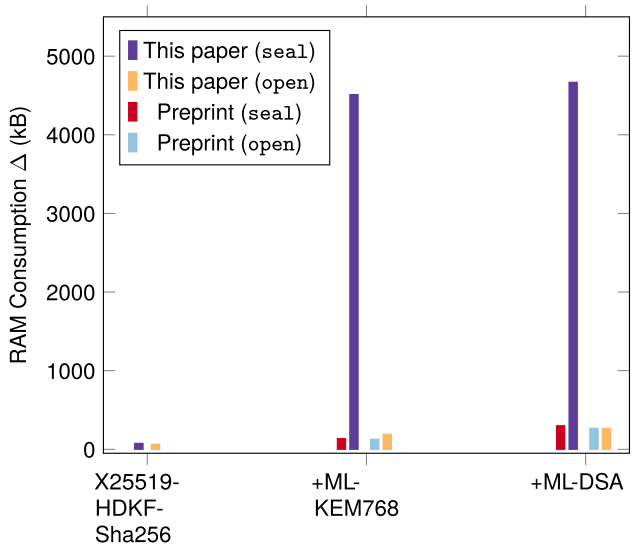

Fig. 12 Memory consumption of HPKE's main operations relative to the dummy ("Empty" implementation. Comparison between preprint version and version from this paper

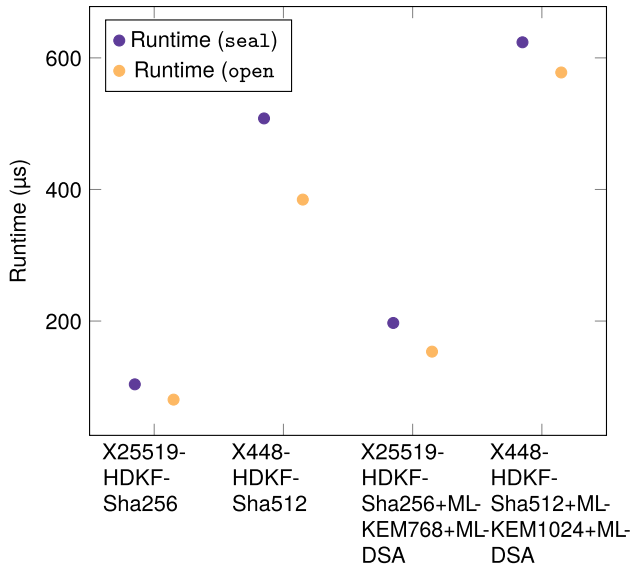

Fig. 13 Run-time of HPKE's main operations with comparison between implementation using level 3 and level 5 primitives using implementations from liboqs

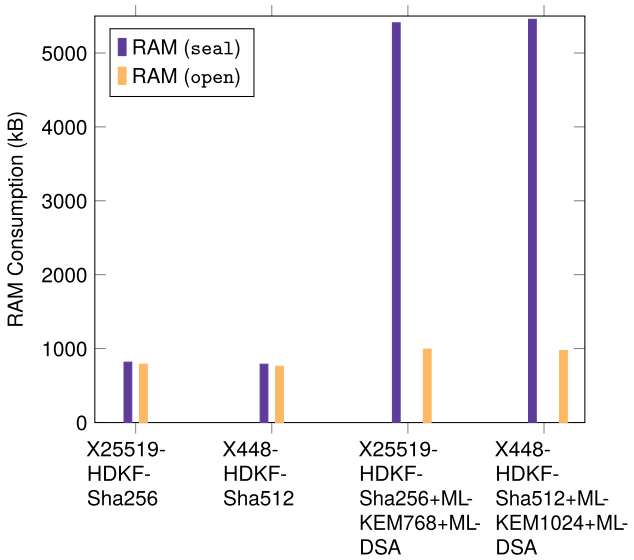

Fig. 14 Memory consumption of HPKE's main operations with comparison between implementation using level 3 and level 5 primitives using implementations from liboqs

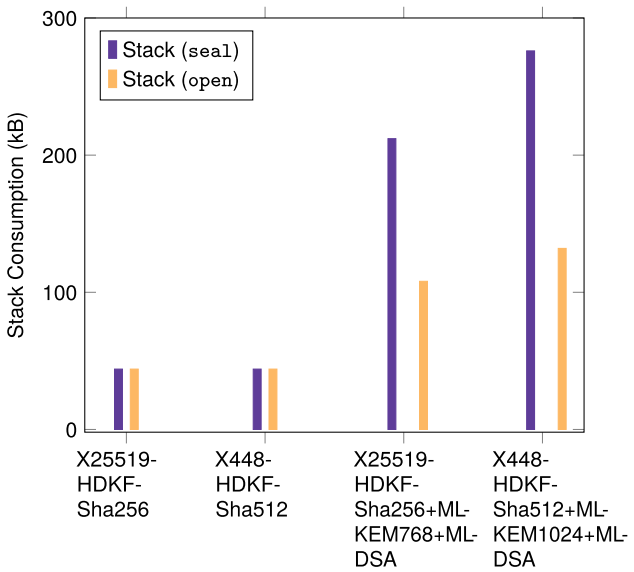

Fig. 15 Stack usage of HPKE's main operations with comparison between implementation using level 3 and level 5 primitives using implementations from liboqs

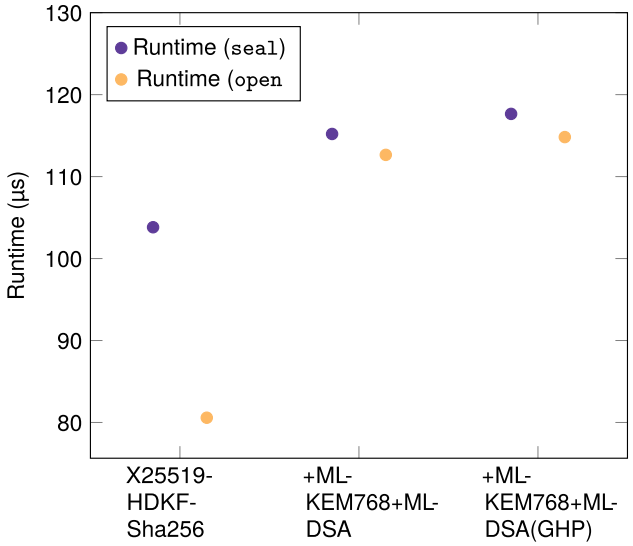

Fig. 16 Run-time of HPKE's main operations with comparison between the combiner proposed in this paper and the generic GHP combiner

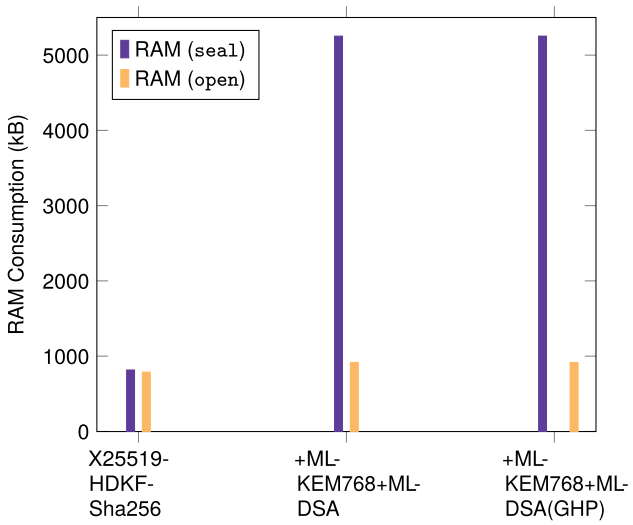

Fig. 17 Memory consumption of HPKE's main operations with comparison between the combiner proposed in this paper and the generic GHP combiner

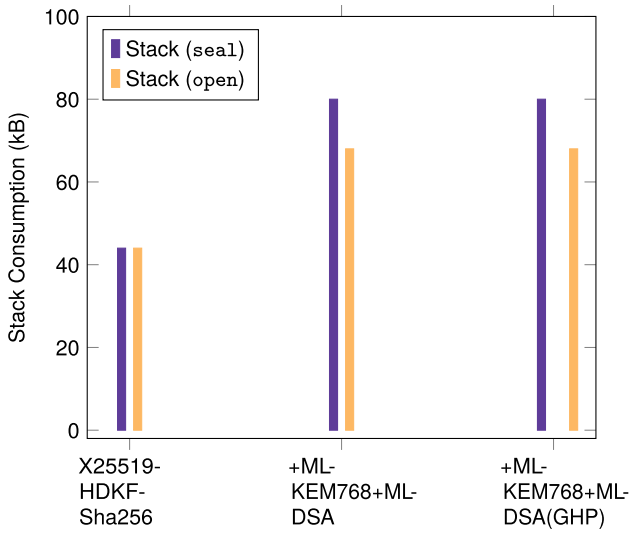

Fig. 18 Stack usage of HPKE's main operations with comparison between the combiner proposed in this paper and the generic GHP combiner

Signatures

Signature algorithms were always available in pre-quantum systems. Pre-quantum and post-quantum variants exist.

```
Signature.keygen() -> (sk, pk)
Signature.sign(sk, pt) -> sig
Signature.validate(pk, sig, pt) -> boolean
Rule:
  Signature.validate(sk, pk) = true
    if sig was generated using Signature.sign() and (sk, pk) was generated using
    ↪  Signature.keygen().
```

KEMs

KEM stands for "Key Encapsulation Mechanism". They can be used to transfer a symmetric key from one party to another; KEMs where introduced as a replacement for NIKEs which are unavailable in the post-quantum setting. Pre-quantum NIKEs and post-quantum NIKEs exist.

```
KEM.keygen() -> (sk, pk)
KEM.encaps(pk) -> (k, ct)
KEM.decaps(sk, ct) -> k or null
Rule:
  KEM.decaps(sk, ct) = k
    if ct was generated using KEM.encaps(pk) and (sk, pk) where generated using
    ↪  KEM.keygen(),
    otherwise null is returned.
```

Authenticated KEMs

AKEMs are a variant of KEMs which provide sender authentication. The interface was introduced in the HPKE standard; pre-quantum variants exist, a post-quantum variant is devised in this paper.

```
AKEM.keygen() -> (sk, pk)
AKEM.encaps(sk1, pk2) -> (k, ct)
AKEM.decaps(sk2, pk1, ct) -> k or null
Rule:
  KEM.decaps(sk1, pk2, ct) = k
    if ct was generated using AKEM.encaps(sk2, pk1) and (sk1, pk1) as well as (sk2, pk2)
    ↪  where generated using KEM.keygen(),
    otherwise null is returned.
```

Appendix 4: X25519MLKEM768

Abstract description of the X25519MLKEM768 cipher in Rust-like pseudocode.

```
const DOMAIN_SEPARATOR_NOAUTH : &str = "Karolin Varner, Wanja Zaeske, Aaron Kaiser, Sven
↪  Friedrich, Alice Bowman, August 2023; From paper: Agile post quantum cryptography in
↪  avionics; AKEM combiner built from AKEM:HPKE/X25519HkdfSha256 + KEM:MLKEM768 +
↪  KDF:shake256: no authentication";
const DOMAIN_SEPARATOR_AUTH : &str = "Karolin Varner, Wanja Zaeske, Aaron Kaiser, Sven
↪  Friedrich, Alice Bowman, August 2023; From paper: Agile post quantum cryptography in
↪  avionics; AKEM combiner built from AKEM:HPKE/X25519HkdfSha256 + KEM:MLKEM768 +
↪  KDF:shake256: authenticated";

fn X25519MLKEM768::encap(sk_mine: Optional<X25519MLKEM768::SecretKey>, pk_theirs:
↪  X25519MLKEM768::PublicKey) {
  // Access the individual subkeys
  let (sks1, sks2) = sk_mine.unwrap_or((None, None)); // Secret key, sender, 1/2
  let (pks1, pks2) = sk_mine.public_key().unwrap_or((None, None)); // Public key, sender,
  ↪  1/2
  let (pkr1, pkr2) = pk_theirs; // Public key, recipient, 1/2

  // Perform encapsulation using kem 1
  let (k1, ct1) = X25519HkdfSha256::encap(sks1, pkr1);
  // Perform encapsulation using kem 2
  let (k2, ct2) = MLKEM768::encap(pkr2);

  // Perform key derivation using a domain separator and both generated keys
  // and concatenate both cipertexts
  let ko = if sk_mine.is_some() {
    Shake256(DOMAIN_SEPARATOR_AUTH || k1 || k2 || ct1 || pks1 || pkr1)[0:256];
  } else {
    Shake256(DOMAIN_SEPARATOR_NOAUTH || k1 || k2 || ct1 || pkr1)[0:256];
  };
  let cto = ct1 || ct2;
  return (ko, cto);
}

fn X25519MLKEM768::decap(sk_mine: X25519MLKEM768::SecretKey, pk_theirs:
↪  Optional<X25519MLKEM768::PublicKey>, ct: X25519MLKEM768::Ciphertext) {
  // Access the individual subkeys and ciphertexts
  let (skr1, skr2) = sk_mine; // Secret key, recipient, 1/2
  let (pkr1, pkr2) = sk_mine.public_key(); // Public key, recipient, 1/2
  let (pks1, pks2) = pk_theirs.unwrap_or((None, None)); // Public key, sender, 1/2

  // Decapsulate the individual ciphertexts
  let k1 = X25519HkdfSha256::decap(skr1, pks1, ct1);
  let k2 = MLKEM768::decap(skr2, ct2);

  // Reconstruct the shared secret by applying the same
  // key derivation step as in the encap function
  let ko = if pk_theirs.is_some() {
    Shake256(DOMAIN_SEPARATOR_AUTH || k1 || k2 || ct1 || pks1 || pkr1)[0:256];
  } else {
    Shake256(DOMAIN_SEPARATOR_NOAUTH || k1 || k2 || ct1 || pkr1)[0:256];
  };

  return ko;
}
```

Appendix 5: X25519MLKEM768MLDSA

Abstract description of the X25519MLKEM768MLDSA cipher in Rust-like pseudocode.

```
const DOMAIN_SEPARATOR_NO_AUTH : &str = "Karolin Varner, Wanja Zaeske, Aaron Kaiser, Sven
↪  Friedrich, Alice Bowman, August 2023; From paper: Agile post quantum cryptography in
↪  avionics; AKEM combiner built from AKEM:HPKE/X25519HkdfSha256 + KEM:ML-KEM768 +
↪  Sig:ML-DSA3 + KDF:shake256: no authentication";
const DOMAIN_SEPARATOR_AUTH : &str = "Karolin Varner, Wanja Zaeske, Aaron Kaiser, Sven
↪  Friedrich, Alice Bowman, August 2023; From paper: Agile post quantum cryptography in
↪  avionics; AKEM combiner built from AKEM:HPKE/X25519HkdfSha256 + KEM:ML-KEM768 +
↪  Sig:ML-DSA3 + KDF:shake256: authenticated";

fn X25519MLKEM768MLDSA::encap(sk_mine: Optional<X25519MLKEM768MLDSA::SecretKey>, pk_theirs:
↪  X25519MLKEM768MLDSA::PublicKey) {
  // Access the individual subkeys
  let (sks1, sks2, sks3) = sk_mine.unwrap_or((None, None, None)); // Secret key, sender,
  ↪  1/2/3
  let (pks1, pks2, pks3) = sk_mine.public_key().unwrap_or((None, None, None)); // Public
  ↪  key, sender, 1/2/3
  let (pkr1, pkr2, pkr3) = pk_theirs; // Public key, recipient, 1/2/3

  // Perform encapsulation using kem 1
  let (k1, ct1) = X25519HkdfSha256::encap(sks1, pkr1);
  // Perform encapsulation using kem 2
  let (k2, ct2) = MLKEM768::encap(pkr2);

  let domain_separator = if sk_mine.is_some() {
    DOMAIN_SEPARATOR_AUTH
  } else {
    DOMAIN_SEPARATOR_NOAUTH
  };

  // Perform key derivation using a domain separator and both generated keys;
  // generate an output key and a key-commitment that can be signed by our Signature.
  let okm = if sk_mine.is_some() {
    Shake256(domain_separator || k1 || k2 || ct1 || pks1 || pkr1)[0:256];

  } else {
    Shake256(domain_separator || k1 || k2 || ct1 || pkr1)[0:256];
  };
  let (ki, kc) = (okm[0:256], okm[256:512]);

  // If no signature key is given, return an empty - zero - signature
  // Else sign the key commitment
  let sig = if sks3 == None {
    [0u8; MLDSA3::SIGNATURE_LENGTH]
  } else {
    MLDSA3::sign(sks3, kc)
  };
  // Encrypt the signature using further output from the key derivation function;
  // this XORs random output from shake256 with the signature
  sig_ct ^= okm[512 : 512 + MLDSA3::SIGNATURE_LENGTH];

  // Second stage of key derivation so we can include the signature in the output key
  let ko = Shake256(domain_separator || "\x01" || ki || sig_ct)[0:256];

  // Concatenate both (A)KEM ciphertexts and the signature
  let cto = ct1 || ct2 || sig_ct;

  return (ko, cto);
}
```

```
fn X25519MLKEM768::decap(sk_mine: X25519MLKEM768::SecretKey, pk_theirs:
↪  Optional<X25519MLKEM768::PublicKey>, ct: X25519MLKEM768::Ciphertext) {
  // Access the individual subkeys and ciphertexts
  let (skr1, skr2, skr3) = sk_mine; // Secret key, recipient, 1/2/3
  let (pkr1, pkr2, pkr3) = sk_mine.public_key(); // Public key, recipient, 1/2/3
  let (pks1, pks2, pks3) = pk_theirs.unwrap_or((None, None, None)); // Public key, sender,
  ↪  1/2/3
  let (ct1, ct2, sig_ct) = ct;

  // Decapsulate the individual ciphertexts
  let k1 = X25519HkdfSha256::decap(skr1, pks1, ct1);
  let k2 = MLKEM768::decap(skr2, ct2);

  let domain_separator = if sk_mine.is_some() {
    DOMAIN_SEPARATOR_AUTH
  } else {
    DOMAIN_SEPARATOR_NOAUTH
  };

  // Reconstruct the shared secret, key commitment, and signature
  // encryption material by applying the same
  // key derivation step as in the encap function
  let okm = if sk_mine.is_some() {
    Shake256(domain_separator || "\x00" || k1 || k2 || ct1 || pks1 || pkr1)[0:256];
  } else {
    Shake256(domain_separator || "\x00" || k1 || k2 || ct1 || pkr1)[0:256];
  };
  let (ki, kc) = (okm[0:256], okm[256:512]);

    // Decrypt the signature
  let sig = sig_ct ^ okm[512 : 512 + MLDSA3::SIGNATURE_LENGTH];

  // If a signature key is given, validate the signature
  if skr3 != None {
    MLDSA3::verify(pks3, sig, kc)
   }

   let ko = Shake256(domain_separator || "\x01" || ki || sig_ct)[0:256];

   return ko;
   }
```

Appendix 6: Encaps and decaps visualized in Drakon

The following two figures (Figs. 19, 20) illustrate the encapsulation/decapsulation operations we introduce in Sect. 4.1. They serve as a more detailed representation of the information conveyed in Table 2.

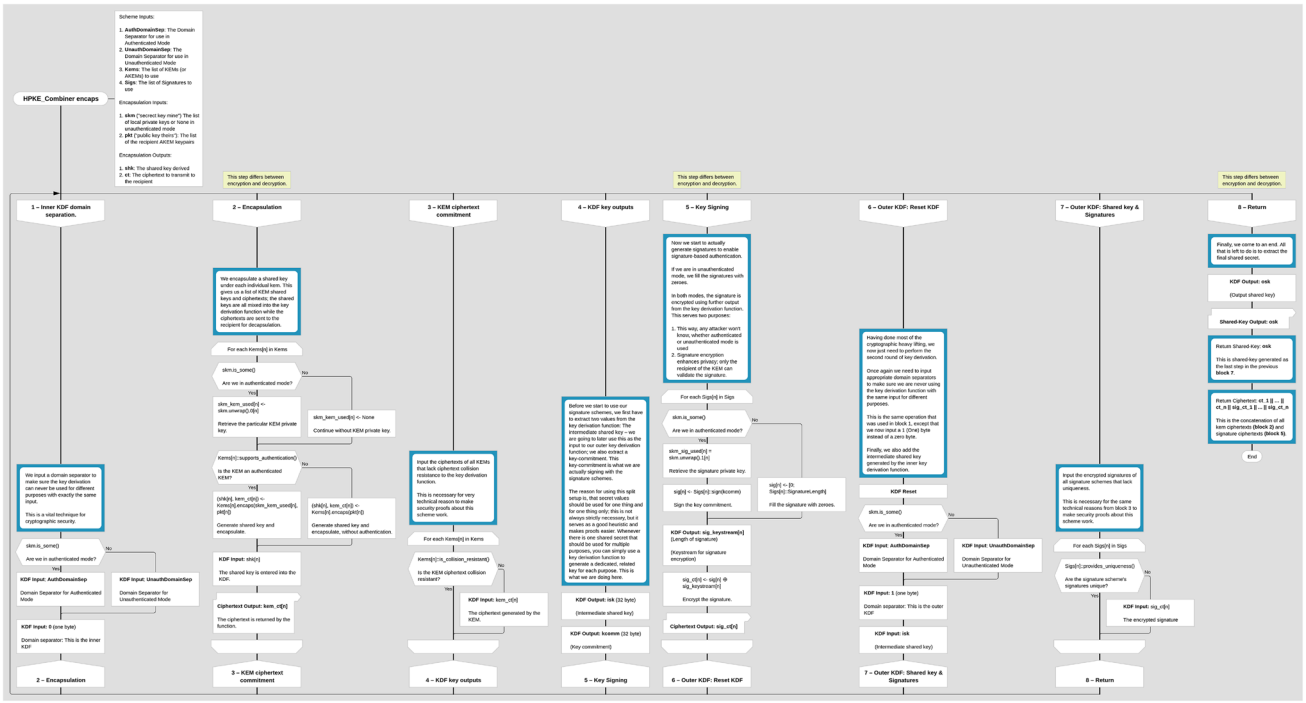

Fig. 19 Encapsulation steps given using the DRAKON [68] visual programming language; created using the Drakon Editor [69]. A bigger version of this chart can be found in the supplemental material (see Sect. 8)

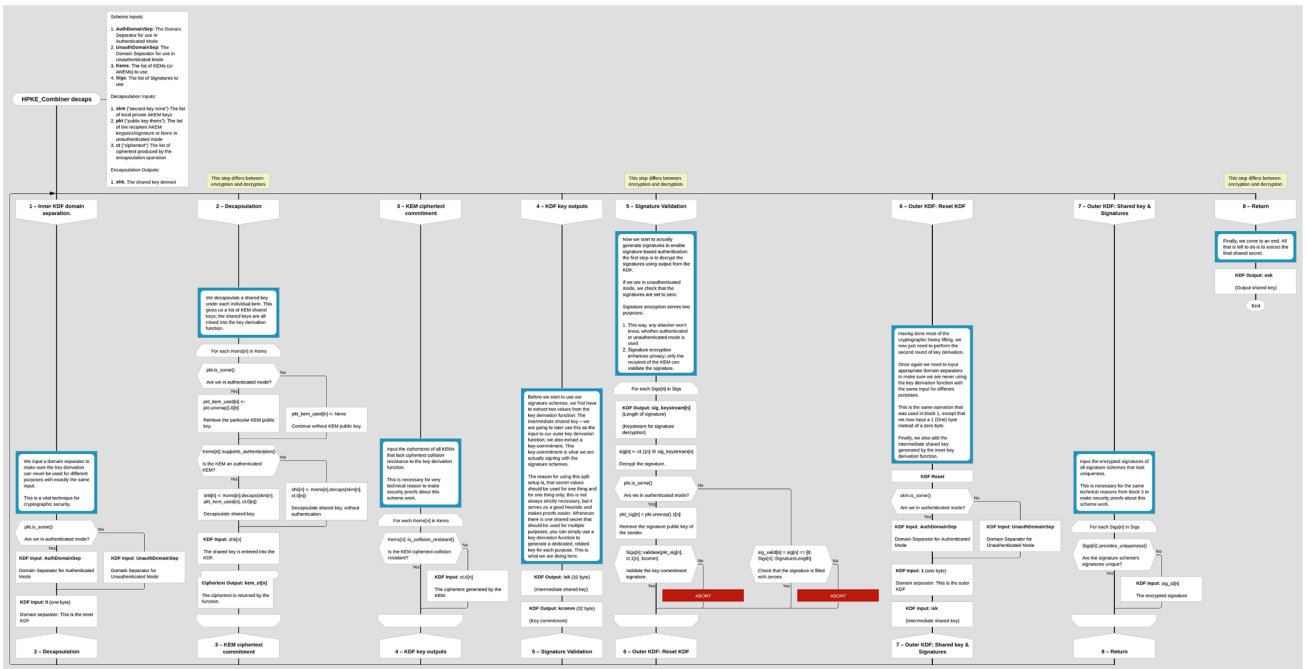

Fig. 20 Decapsulation steps given using the DRAKON [68] visual programming language; created using the Drakon Editor [69]. A bigger version of this chart can be found in the supplemental material (see Sect. 8)

Acknowledgements We would like to thank Bas Westerbaan and Douglas Stebila for their work on hybridizing x25519 and Kyber [63] for HPKE; while our scheme was independently developed, we did use their implementation as a basis for ours.

Author contributions K. V. Cryptography; post-quantum encryption design, project lead, writing lead on Introduction, Cryptography, The basics of encryption, Engineering encryption systems: A checklist, Off-the-shelf encryption systems, Post-quantum cryptography, Migrating encryption protocols to post-quantum security, Our Contribution, Post-quantum security for HPKE, Evaluation: Security, Outlook, Conclusion. W. Z. Avionics; partition system design, writing lead on Avionics Software Engineering, Certification in Avionics, Comparing Avionics and Cryptography, Related works, Our Contribution, Integrating HPKE in an ARINC 653 partition. S. F. Performance evaluation; writing lead on Evaluation: Performance and Memory Overhead. A. K. Analysis of security; accounting for attacks based on ciphertext collision. A. B. Lead editor. Lisa Schmidt Graphical design (poster at DLRK conference).

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability While not primarily focused on data, for this study we benchmarked various variants of our HPKE extensions. The implementation thereof, the bench-harness used and our measurements are to be found in the following GitHub repository: https://github.com/rosenpass/paper-hpke-in-avionics-supplemental.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Smith, M., Moser, D., Strohmeier, M., Lenders, V., Martinovic, I.: Economy class crypto: exploring weak cipher usage in avionic communications via ACARS (2017)
2. Schäfer, M., Lenders, V., Martinovic, I.: Experimental analysis of attacks on next generation air traffic communication. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) Applied Cryptography and Network Security, pp. 253–271. Springer, Berlin (2013) . (**isbn:978-3-642-38980-1**)
3. RTCA: DO-178C software considerations in airborne systems and equipment certification. Standard. RTCA (2011)
4. RTCA: DO-297 Integrated Modular Avionics (IMA) development guidance and certification considerations. Standard
5. ARINC: Avionics application software standard interface part 0 overview of ARINC 653. Standard. Aeronautical Radio Incorporated, Bowie (2019)
6. Friedrich, S., Engler, E., Schubert, T., Zaeske, W., Durak, U.: Assuring APEX with a versatile Rust API. In embedded world 2023 Conference Proceedings, pp. 298–305. WEKA FACHMEDIEN GmbH (2023) (**isbn: 978-3-645-50197-2**)
7. Cabler, S.J.M.: Reusable software components. AC 20-148. (2004). https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_20-148.pdf
8. Gentry, C.: A Fully Homomorphic Encryption Scheme. Stanford University, Stanford (2009)
9. Falkner, S., Kieseberg, P., Simos, D.E., Traxler, C., Weippl, E.: E-voting authentication with QR-codes. In: Human Aspects of Information Security, Privacy, and Trust: Second International Conference, HAS: Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22–27, 2014. Proceedings 2, pp. 149–159. Springer (2014)
10. Goldreich, O.: Secure multi-party computation. Manuscr. Prelim. Vers. **78**, 110 (1998)
11. Davis, A., Rubinstein, M., Wadhwa, N., Gautham, J.M., Durand, F., Freeman, W.T.: Passive recovery of sound from video, the visual microphone (2014)
12. Politician's fingerprint reproduced using photos of her hands. https://arstechnica.com/information-technology/2014/12/politicians-fingerprint-reproduced-using-photos-of-her-hands/. Archived: https://web.archive.org/web/20230904163754/https://arstechnica.com/information-technology/2014/12/politicians-fingerprint-reproduced-using-photos-of-her-hands/
13. Nassi, B., Iluz, E., Cohen, O., Vayner, O., Nassi, D., Zadov, B., Elovici, Y.: Video-based cryptanalysis: extracting cryptographic keys from video footage of a device's power LED. Cryptology ePrint Archive (2023)
14. Clark, S., Goodspeed, T., Metzger, P., Wasserman, Z., Kevin, X., Blaze, M.: Why (special agent) Johnny (still) can't encrypt: A security analysis of the APCO Project 25 two-way radio system. In: USENIX Security Symposium. 2011: 8–12 (2011)
15. Varner, K.: Decryption despite errors. https://github.com/koraa/decryption-despite-errors/blob/main/paper/decryption-despite-errors.pdf
16. Von Mises, R.: Über aufteilungs-und besetzungswahrscheinlichkeiten (1939)
17. Salowey, J.A., McGrew, D., Choudhury, A.: AES Galois Counter Mode (GCM) cipher suites for TLS. RFC 5288. (2008). https://doi.org/10.17487/RFC5288
18. Harnik, D., Kilian, J., Naor, M., Reingold, O., Rosen, A.: On robust combiners for oblivious transfer and other primitives. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 96–113. Springer (2005)
19. Barnes, R., Bhargavan, K., Lipp, B., Wood, C.A.: Hybrid public key encryption. RFC 9180 (2022). https://doi.org/10.17487/RFC9180
20. Rescorla, E.: The transport layer security (TLS) protocol version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446
21. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The messaging layer security (MLS) protocol. RFC 9420 (2023). https://doi.org/10.17487/RFC9420
22. Freier, A.O., Karlton, P., Kocher, P.C.: The secure sockets layer (SSL) protocol version 3.0. RFC 6101 (2011). https://doi.org/10.17487/RFC6101
23. Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., Cooper, D.: Internet X.509 public key infrastructure certificate and Certificate Revocation List (CRL) profile. RFC 5280 (2008). https://doi.org/10.17487/RFC5280
24. Bhargavan, K., Brzuska, C., Fournet, C., Green, M., Kohlweiss, M., Zanella-Béguelin, S.: Downgrade resilience in

key-exchange protocols. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 506–525. (2016). https://doi.org/10.1109/SP.2016.37
25. Alashwali, E.S., Rasmussen, K.: What's in a downgrade? A taxonomy of downgrade attacks in the TLS protocol and application protocols using TLS. In: Security and Privacy in Communication Networks: 14th International Conference, SecureComm: Singapore, August 8–10, 2018, Proceedings. Part II, vol. 2018, pp. 468–487. Springer, Singapore (2018)
26. Schwabe, P., Stebila, D., Wiggers, T.: Post-quantum TLS without handshake signatures. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. CCS '20. Association for Computing Machinery, Virtual Event, USA, pp. 1461–1480 (2020). https://doi.org/10.1145/3372297.3423350 (**isbn: 9781450370899**)
27. Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.-Y., Zanella-Béguelin, S.: Proving the TLS handshake secure (as it is). In Advances in Cryptology-CRYPTO,: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2014, Proceedings, Part II 34, vol. 2014, pp. 235–255. Springer (2014)
28. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking post-quantum cryptography in TLS. In: Post-Quantum Cryptography: 11th International Conference, PQCrypto: Paris, France, April 15–17, 2020, Proceedings 11, vol. 2020, pp. 72–91. Springer (2020)
29. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Post-quantum authentication in TLS 1.3: a performance study. Cryptology ePrint Archive (2020)
30. Gonzalez, R., Wiggers, T.: KEMTLS vs. post-quantum TLS: performance on embedded systems. In: International Conference on Security, Privacy, and Applied Cryptography Engineering, pp. 99–117. Springer (2022)
31. Perrin, T.: The Noise protocol framework (2016). http://noiseprotocol.org/noise.pdf
32. Koblitz, N., Menezes, A., Vanstone, S.: The state of elliptic curve cryptography. Des. Codes Cryptogr. **19**, 173–193 (2000)
33. Marlinspike, M., Perrin, T.: The x3dh key agreement protocol. Open Whisp. Syst. **283**, 10 (2016)
34. Yanofsky, N.S., Mannucci, M.A.: Quantum Computing for Computer Scientists. Cambridge University Press, Cambridge (2008)
35. National Institute of Standards an Technology (NIST). NIST announces first four quantum-resistant cryptographic algorithms (2022). https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms
36. McEliece, R.J.: A public-key cryptosystem based on algebraic. Coding Thv **4244**, 114–116 (1978)
37. Bernstein, D.J., Chou, T., Lange, T., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., et al.: Classic McEliece (2017)
38. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M. Schwabe, P., Seiler, G., Stehlé, D. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In: IEEE European Symposium on Security and Privacy (EuroS &P), vol. 2018, pp. 353–367. IEEE (2018)
39. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehl é, D.: Crystals-dilithium: a lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 238–268 (2018)
40. Prest, T., Fouque, P.-A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Whyte, W., Zhang, Z., Falcon.: Post-Quantum Cryptography Project of NIST, Gregor Seiler (2020)
41. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS+ signature framework. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, pp. 2129–2146 (2019)
42. NIST Fips Pub: 203: Module-lattice-based key-encapsulation mechanism standard. Fed. Inf. Process. Stand. Publ. **203**(17956), 66052–66057 (2024)
43. NIST Fips Pub: 204: module-lattice-based digital signature standard. Fed. Inf. Process. Stand. Publ. **204**(17956), 66052–66057 (2024)
44. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
45. Blanchet, B.: Symbolic and computational mechanized verification of the ARINC823 avionic protocols. In: 2017 IEEE 30th Computer Security Foundations Symposium (CSF), pp. 68–82. (2017). https://doi.org/10.1109/CSF.2017.7
46. Mäurer, N., Gräupl, T., Schmitt, C., Rodosek, G.D., Reiser, H.: Advancing the security of LDACS. EEE Trans. Netw. Serv. Manag. **19**(4), 5237–5251 (2022). https://doi.org/10.1109/TNSM.2022.3189736
47. Deutsches Zentrum fuer Luft- und Raumfahrt e. V. LDACS implementation and validation. https://www.ldacs.com/ldacs1-development/ldacs-implementation-and-validation/. Accessed 26 July 2023 (2023)
48. Boegl, T., Rautenberg, M., Haindl, B., Rihacek, C., Meser, J., Fantappie, P., Pringvanich, N., Micallef, J., Hauf, K., MacBride, J., Sacre, P., van den Einden, B., Gräupl, T., Schnell, M.: LDACS white paper—a roll-out scenario. WORKING PAPER. ICAO (2019)
49. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. https://eprint.iacr.org/2022/975 (2022)
50. EUROCONTROL: aeronautical mobile airport communications system datalink. https://www.eurocontrol.int/system/aeronautical-mobile-airport-communications-system-datalink. Accessed 27 July 2023. (2023)
51. WiMAX Forum: Aeromacs. https://wimaxforum.org/Page/AeroMACS. Accessed 27 July 2023. (2023)
52. WiMAX Forum: WiMAX Forum Security
53. The Open Quantum Safe Project. X.509 | Open Quantum Safe. https://openquantumsafe.org/applications/x509.html. Accessed 11 Sept 2023. (2023)
54. Morris J Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions (2015)
55. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Annual Cryptology Conference, pp. 631–648. Springer (2010)
56. Barbosa, M., Connolly, D., Duarte, J.D., Kaiser, A., Schwabe, P., Varner, K., Westerbaan, B.: X-wing: the hybrid KEM you've been looking for. Cryptology ePrint Archive, Paper 2024/039. https://eprint.iacr.org/2024/039 (2024)
57. Giacon, F., Heuer, F., Poettering, B.: KEM combiners. In: Abdalla, M., Dahab, R. (eds.) Public-Key Cryptography–PKC 2018, pp. 190–218. Springer International Publishing, Cham (2018) . (**ISBN: 978-3-319-76578-5**)
58. Goldwasser, S., Bellare, M.: Lecture notes on cryptography. Summer course "Cryptography and computer security", p. 1999. MIT (1996)
59. Morgan, A., Pass, R.: On the security loss of unique signatures. In: Theory of Cryptography Conference, pp. 507–536. Springer (2018)
60. Bellare, M., Davis, H., Günther, F.: Separate your domains: NIST PQC KEMs, oracle cloning and read-only in differentiability. In: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30, pp. 3–32. Springer, (2020)
61. Stebila, D., Mosca, M.: Post-quantum key exchange for the internet and the open quantum safe project. In: International

Conference on Selected Areas in Cryptography, pp. 14–37. Springer (2016)
62. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In: Advances in Cryptology—CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27: Proceedings 18, vol. 1998, pp. 26–45. Springer (1998)
63. Westerbaan, B., Wood, C.A.: X25519Kyber768Draft00 hybrid post-quantum KEM for HPKE. Internet-Draft draft-westerbaan-cfrg-hpke-xyber768d00-03. Work in Progress. Internet Engineering Task Force, May 2024. https://datatracker.ietf.org/doc/draft-westerbaan-cfrg-hpke-xyber768d00/03/
64. Varner, K., Zaeske, W., Friedrich, S., Kaiser, A., Bowman, A.: Agile, post-quantum secure cryptography in avionics (preprint from,: Cryptology ePrint Archive, Paper 2024/667. Version **20240501**(143223) (2024)
65. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: NIST special publication 800-57. NIST Spec. Publ. **800**(57), 1–142 (2007)
66. NIST Fips Pub: 197: Advanced encryption standard (AES). Fed. Inf. Process. Stand. Publ. **197**(441), 0311 (2001)
67. Langley, A., Hamburg, M., Turner, S.: RFC 7748: elliptic curves for security (2016)
68. Parondzhanov, V.D.: Visual syntax of the DRAKON language. Programming And Computer Software (Official English Translation of Programmirovanie) 21.3, pp. 142–153 (1995). https://drakon.su/_media/video_i_prezentacii/graphical_syntax_.pdf (**issn: 0361-7688**)
69. Mitkin, S.: Drakonhub Desktop. https://github.com/stepan-mitkin/drakonhub_desktop (2024)