AGILE, POST-QUANTUM SECURE CRYPTOGRAPHY IN AVIONICS

Karolin Varner[†‡], Wanja Zaeske[*‡], Sven Friedrich[*], Aaron Kaiser[†], Alice Bowman[‡]

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

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

[‡] Rosenpass e. V. Postfach 3212, 30032 Hannover

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 (section 5) constitute initial evidence that both the memory and performance characteristics of this approach are suitable for deployment in flight-computers currently in use. Prior to optimisation, performance measurements show a modest memory requirement of under 400 KB of RAM, but employ a more substantial stack usage of just under 200 KB. Our most advanced redundant post-quantum cipher is five times slower than its non-redundant, pre-quantum counterpart.

Keywords

Avionics; Crypto-Agility; Post-Quantum Cryptography; Robust Combiners; HPKE; Kyber; Dilithium

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 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. 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 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 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 & 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

[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

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

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 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 for future aircraft type certifications.

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

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. prequantum, 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 violenty 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 four 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].

[4]Technically these asymmetric post-quantum schemes are not ciphers; 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.

Cryptography considers passive – eavesdropping – attackers and active attackers: those who intercept messages and try to manipulate them. Using Kyber without an interactive protocol, provides passive post-quantum security. Security against active attackers requires either a combination of Kyber and a signature scheme or an interactive[26, 42] protocol. In this mode, the sender proves their identity by decrypting a challenge-ciphertext.

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.

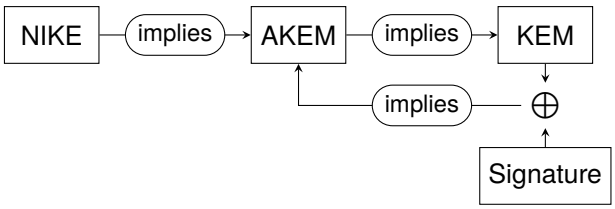

FIG 1. NIKE is the strongest asymmetric interface; a NIKE can do anything a KEM or an AKEM can do. The novel AKEMs are closer to NIKEs than KEMs. An AKEM can be built from a NIKE or from a KEM and a Signature scheme. "Implies" in this context is a technical term from mathematics; you can read "NIKE implies 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 (Figure 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 A 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[43] 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 Method"
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 Method"
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 algorithms 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 authenticity guard, and most aviation radio frequency protocols currently used do not feature effective measures to implement confidentiality.

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.

3. RELATED WORKS

Previously in section 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 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) [44]. While performing a cryptographic analysis of AMS, Blanchet found problems with it, which were communicated back to the industry editor of ARINC 823 [44]. Still, since the publication of [44] 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 mono-alphabetic substitution cipher, which can only be described as security theatre [1].

TAB 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
Kyber512	800	1632	768
Kyber512-X25519	832	1664	800

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. [45–47]

One goal of LDACS is the establishment of security at the link layer [47], 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 [45]. 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) [45]. Unfortunately, Castryck and Decru published a critical vulnerability in SIKE [48] approximately one year after the publication of the LDACS Cell Attachment paper [45]. 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.

Analysis of the proposed protocol, in combination with the specific selection of available cipher suites, allowed the author of [45] 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 [49, 50]. 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 [45, 51]. As Mäurer, Gräupl, Schmitt, Rodosek, and Reiser point out, AeroMACS "only supports one cipher suite option" [45], 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 [45]. X.509 certificates are compatible with post-quantum cryptography [52]. 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.

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 of demonstration of crypto-agility, by swapping the cryptography in the demonstrator.

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.

HPKE[19] is a fairly recent standard for asymmetric encryption that provides both authenticity and secrecy for asynchronous one way communication (section 2.6, section 2.8). The standard uses KEMs "Key Encapsulation Methods" and AKEMs – "Authenticated KEM", a type of KEM where the sender needs to authorise themselves in order to be allowed to transmit any data.

TAB 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 Kyber, providing additional, redundant post-quantum secrecy. The third column shows the HPKE standard construction plus both Kyber and Dilithium, providing the full set of redundant post-quantum security properties: Secrecy and authenticity.

	HPKE Standard	+Kyber	+Kyber +Dilithium
Secrecy			
Classical	✓	✓	✓
Post-Quantum	✗	✓	✓
Authenticity			
Classical	✓	✓	✓
Post-Quantum	✗	✗	✓

Both constructions given in this paper use the SHA-3[53] 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 section 6.

To build X25519Kyber768, we add Kyber[38] a key-derivation-function[54] ("KDF") based combiner, instantiated with shake256[53] as a key derivation function. That is, we use both key encapsulation methods 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[5] and as a measure of heuristic security, we also include the X25519HkdfSha256 public keys.

[5] X25519HkdfSha256 does not provide ciphertext collision resistance[55]; 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[56]. 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

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 B for a detailed description of the X25519Kyber768 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 *Dilithium3*[39] 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.

This construction is not quite as straightforward as the previous one as giving an attacker access to a signature breaks anonymity[57]. 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 section 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 Kyber768) provide anonymity and secrecy; if one component is giving away the sender identity, hiding a different component cannot solve this. See Appendix C for the full definition of the X25519Kyber768Dilithium construction.

The variant X25519Kyber768 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 Kyber ensures the secrecy of data even in the presence of an quantum computer. The variant X25519Kyber768Dilithium should be used once cryptographicaly relevant quantum computers exist, as X25519 can no longer be used to ensure authenticity of the data and therefore Dilithium 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[55] and which of the signatures provide uniqueness[58].

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 section 6).

The construction follows the recipes for encapsulation and decapsulation listed in Table 3.

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 (section 2.8) for use within HPKE.
See section 2.6 and section 2.8.

X25519HkdfSha256[19] The pre-quantum KEM
All constructions studied in the work combine two AKEMs (section 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[53] 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.

Kyber768[38] The post-quantum KEM
Kyber is the standardized post-quantum secure key transferral method. It comes in three variants, and we chose the middle variant as a balance between speed and security.
Kyber is what endows our constructions with post-quantum secrecy.

Dilithium3[39] The post-quantum signature
Dilithium is one of the standardised post-quantum signatures. Dilithium3 is the middle variant again:

mathematical model of KEMs. Given the small impact of hashing 32 additional bytes of data, we prefer to stick to the established models.

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.

4.2. Integrating HPKE in an ARINC 653 partition

As outlined in section 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 possilbe that only a *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 section 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. 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 are exchanged via sampling ports, a directed flavour of shared memory between partitions.

Figure 2 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 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 consumption across its various configurations. Both values are important in an avionics context as partitions are allocated discrete amounts of computation time, and memory allowance,

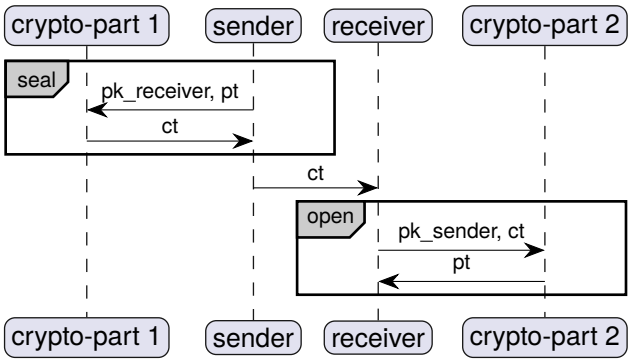

FIG 2. Interaction between the three partition in the demonstrator

TAB 3. The steps to construct a new KEM or AKEM with redundancy using our recipe. If no signature scheme is used or if all the signature schemes used provide uniqueness[58], 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[59].

	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 KDF:	Input a unique domain separator[59] and all the decapsulated secrets into the KDF. Include the ciphertexts of any constituent KEMs that lack ciphertext collision resistance[55].	
3 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	
2 KDF:	Input a unique domain separator[59] and the intermediate key. Include the signatures of any constituent signature schemes that lack uniqueness[58].	
3 KDF (output key):	Extract a 32 byte output key – the shared secret – from the key derivation function	
8 Return:	The output key, each KEM ciphertext, and each encrypted signature	The output key

which cannot be exceeded. Our measurements were conducted on an AMD EPYC 7542 processor, running Linux 6.1.64, without an ARINC 653 execution environment. We omitted ARINC 653, when benchmarking, as we found it had a negligible impact on memory and runtime overhead. It also hindered the measurement due to isolation and its static time-slicing scheduler. The processor was chosen for its comparatively low and stable base clock of 2.9 GHz. To determine the memory consumption for the runtime performance evaluation citerion.rs[6], the `softlimit` utility from the daemontools [7] were used. Using `softlimit`'s `-a` switch, the minimum acceptable amount of memory for each variant of HPKE to `seal`/`open` a message was determined. Similarly, using the `-s` switch we measured the minimum acceptable stack size. We recognise that both measurements heavily depend on the hardware, especially the processor used, and operating system in use. Despite this, we are able to use this setup to paint a qualitative picture of whether or not post-quantum adaption is infeasible as a result of runtime overhead.

[6] https://github.com/bheisler/criterion.rs
[7] http://cr.yp.to/daemontools.html

The results of the measurements on runtime (Figure 3) suggest that there is only a moderate increase from pre-quantum to post-quantum hybrid HPKE (`HKDF-Sha256+Kyber768`) of approximately 31 %. Adding Dilithium affects the encryption and signing (`seal`) significantly more at approximately 242 % percent increased runtime compared to `HKDF-Sha256+Kyber768`. Decryption and signature verification (`open`), however, is only about 163 % slower. The `seal` measurement for `HKDF-Sha256+Kyber768+Dilithium` was the only runtime measurement with notable deviation (hence it is the only measurement with error bars). Comparing the worst outliers, moving from pre-quantum to hybrid post-quantum security in HPKE incurs a worst-case performance penalty of roughly a factor of four, which is acceptable for many use-cases.

Considering the minimum required amount of memory (Figure 4 and Figure 5), the overhead seems even more modest. The increase in total memory required from `HKDF-Sha256` to `HKDF-Sha256+Kyber768` is approximately 7 % relative or at most 68 kB for both

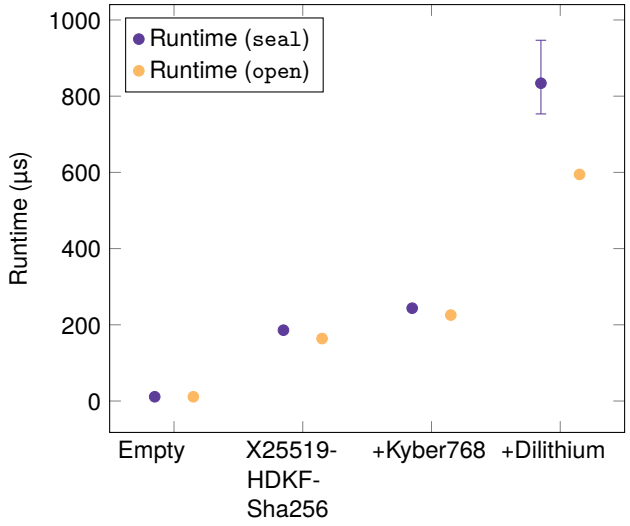

FIG 3. Run-time of HPKE's main operations. Due to variance in the measurement for `seal` with Dilithium, a 95% quantile error bar is included

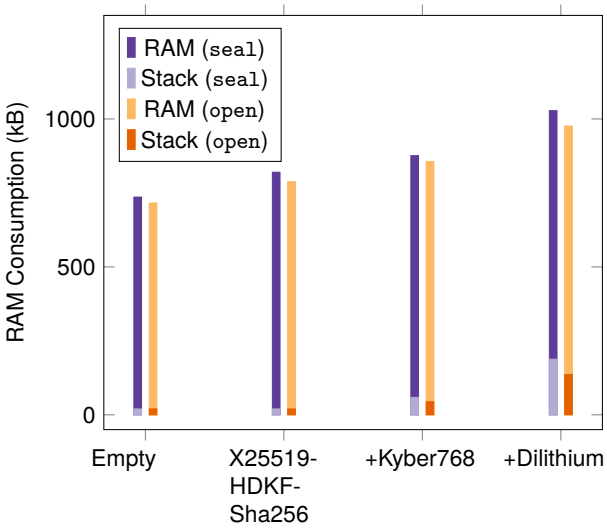

FIG 4. Memory consumption of HPKE's main operations. RAM measures the total memory required (including machine code, static sections, etc.), while *Stack* only refers to the minimum allowable stack size for execution

`seal` and `open`. Adding Dilithium to HPKE bumps the peak memory consumption by another 17 % or 182 kB at most. Notable, however, is the minimum allowable stack size, which rises measurably with the addition of Kyber768 and Dilithium respectively. Most importantly, compared to `HKDF-Sha256` the stack size has to be increased from 20 kB to 188 kB for `HKDF-Sha256+Kyber768+Dilithium`. For small embedded systems, which only feature a couple hundred kB of RAM, this 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 2000s. We therefore conclude that utilisation of our fully post-quantum secure HPKE variant in current avionics hardware is neither prohibited by either the memory nor runtime overhead.

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's 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 X25519Kyber768 and X25519Kyber768Dilithium 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 (Kyber and Dilithium) and pre-quantum schemes (X25519HkdfSha256). To show security in the pre-quantum setting we can not rely on the security of Kyber and Dilithium; to show

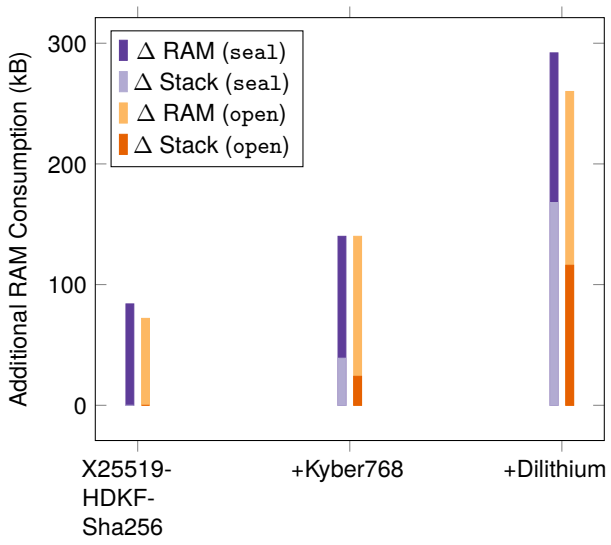

FIG 5. Memory consumption of HPKE's main operations, relative to Empty. RAM measures the additional memory required (including machine code, static sections, etc.), while *Stack* only refers to the increase in minimum allowable stack size for execution both compared to the empty case from Figure 4

our scheme is secure in the post-quantum setting, we cannot rely on X25519HkdfSha256.

The basic scenarios we must consider when analysing our scheme.

X25519Kyber768, pre-quantum
The scheme should provide all security properties (secrecy, authenticity, and identity hiding) against active adversaries without relying on Kyber for security.

X25519Kyber768, post-quantum
The scheme should provide secrecy and identity hiding, but not authenticity against active adversaries without relying on X25519HkdfSha256.

X25519Kyber768Dilithium, pre-quantum
The scheme should provide all security properties without relying on Kyber for security.

X25519Kyber768, 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 X25519Kyber768 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[8]. Then the adversary 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 Kyber, the adversary is not able to obtain the shared secret generated by Kyber and therefore cannot compute the shared secret of the composite AKEM. In the post-quantum case, X25519Kyber768 does not provide authenticity.

The security arguments for X25519Kyber768Dilithium are similar to the arguments for X25519Kyber768. 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 Dilithium 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 Kyber768 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 Dilithium 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 Dilithium 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[56] figured out that the IND-CCA[9] 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[55] with hybrid security where it was

[8] Possibly gained by eavesdropping on some legitimate key transferral session.

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

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[60].

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 X25519Kyber768 ciphertext has two fields: The ciphertext belonging to X25519HkdfSha256 and the one belonging to Kyber. The X25519Kyber768Dilithium ciphertext has one additional field, the encrypted dilithium 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.

Kyber provides ciphertext collision resistance, according to the X-Wing analysis[55], so this field cannot lead to a collision.

The Dilithium signature is slightly more complex. Dilithium does not provide uniqueness[58], but we encrypt the signature under key derived from both X25519HkdfSha256 and Kyber 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 Dilithium, 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 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[10]. 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 Kyber'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 section 2.6) asymmetric encryption standard in an ARINC 653 partition (section 4.2).

In order to ensure that the solution provides post-quantum security (see section 2.7), we devised two quantum resistant authenticated key-transferral (see section 2.8) schemes (section 4.1). One scheme uses the standardised, quantum-resistant key-transferral scheme Kyber to provide security against passive attackers. The other scheme combines Kyber with the standardised post-quantum signature scheme Dilithium to provide security against active attackers. (See section 2.7 on passive/active attackers). All of our quantum-resistant ciphers integrate a pre-quantum cipher to provide for cryptographic redundancy.

Performance measurements (section 5) provide initial evidence that the memory-requirements and performance characteristics may be suitable for deployment in the flight-computers already being used. On our strongest, post-quantum secure cipher, peak RAM usage was below 400KB, though stack usage was high

[10] "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.

for some measurements at just below 200KB. Performance measurements are hard to give in absolute numbers. Nonetheless, all operations finished in under a millisecond on 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 flight computers in use today are suitable for the task. Similarly, software 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 400KB of additional RAM, but does come with a large stack size requirement of just under 200KB. Stack size is a prevailing issue with post-quantum ciphers, with our strongest post-quantum variant having a forty times higher stack size requirement than the pre-quantum variant. 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 five 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. Flight 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.

Contact address:

wanja.zaeske@dlr.de

Attribution:

Karolin Varner 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.
Wanja Zaeske 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.
Sven Friedrich Performance evaluation; writing lead on Evaluation: Performance and Memory Overhead.
Aaron Kaiser Analysis of security; accounting for attacks based on ciphertext collision.
Alice Bowman Lead editor.
Lisa Schmidt Graphical design (poster at DLRK conference).

References

[1] Matthew Smith, Daniel Moser, Martin Strohmeier, Vincent Lenders, and Ivan Martinovic. Economy class crypto: Exploring weak cipher usage in avionic communications via acars. In. Apr. 2017.

[2] Matthias Schäfer, Vincent Lenders, and Ivan Martinovic. Experimental analysis of attacks on next generation air traffic communication. In *Applied Cryptography and Network Security*. Ed. by Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 253–271. 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. RTCA, 2005.

[5] ARINC. *Avionics Application Software Standard Interface part 0 overview of arinc 653*. Standard. Bowie, MD, USA: Aeronautical Radio Incorporated, Aug. 2019.

[6] Sven Friedrich, Emil Engler, Tim Schubert, Wanja Zaeske, and Umut Durak. Assuring apex with a versatile rust api. In *embedded world 2023 Conference Proceedings*. WEKA FACHMEDIEN GmbH, Mar. 2023, pp. 298–305. ISBN: 978-3-645-50197-2.

[7] Susan J.M. Cabler. *The transport layer security (tls) protocol version 1.3*. AC 20-148. Dec. 2004. https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_20-148.pdf.

[8] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[9] Stefanie Falkner, Peter Kieseberg, Dimitris E Simos, Christina Traxler, and Edgar Weippl. E-voting authentication with qr-codes. In *Human Aspects of Information Security, Privacy, and Trust: Second International Conference, HAS 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings 2*. Springer. 2014, pp. 149–159.

[10] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version* 78.110 (1998).

[11] Abe Davis, Michael Rubinstein, Neal Wadhwa, Gautham J Mysore, Fredo Durand, and William T Freeman. The visual microphone: Passive recovery of sound from video (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] Ben Nassi, Etay Iluz, Or Cohen, Ofek Vayner, Dudi Nassi, Boris Zadov, and Yuval Elovici. Video-based cryptanalysis: Extracting cryptographic keys from video footage of a device's power led. *Cryptology ePrint Archive* (2023).

[14] Sandy Clark, Travis Goodspeed, Perry Metzger, Zachary Wasserman, Kevin Xu, and Matt Blaze. Why (special agent) johnny (still) can't encrypt: A security analysis of the apco project 25 two-way radio system. In *USENIX Security Symposium*. Vol. 2011. 2011, pp. 8–12.

[15] Karolin Varner. *Decryption despite errors*. https://github.com/koraa/decryption-despite-errors/blob/main/paper/decryption-despite-errors.pdf.

[16] Richard Von Mises. *Über aufteilungs-und besetzungswahrscheinlichkeiten*. na, 1939.

[17] Joseph A. Salowey, David McGrew, and Abhijit Choudhury. *Aes galois counter mode (gcm) cipher suites for tls*. RFC 5288. Aug. 2008. DOI: 10.17487/RFC5288. https://www.rfc-editor.org/info/rfc5288.

[18] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2005, pp. 96–113.

[19] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. *Hybrid public key encryption*. RFC 9180. Feb. 2022. DOI: 10.17487/RFC9180. https://www.rfc-editor.org/info/rfc9180.

[20] Eric Rescorla. *The transport layer security (tls) protocol version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. https://www.rfc-editor.org/info/rfc8446.

[21] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. *The messaging layer security (mls) protocol*. RFC 9420. July 2023. DOI: 10.17487/RFC9420. https://www.rfc-editor.org/info/rfc9420.

[22] Alan O. Freier, Philip Karlton, and Paul C. Kocher. *The secure sockets layer (ssl) protocol version 3.0*. RFC 6101. Aug. 2011. DOI: 10.17487/RFC6101. https://www.rfc-editor.org/info/rfc6101.

[23] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. *Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile*. RFC 5280. May 2008. DOI: 10.17487/RFC5280. https://www.rfc-editor.org/info/rfc5280.

[24] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss, and Santiago Zanella-Béguelin. Downgrade resilience in key-exchange protocols. In *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 506–525. DOI: 10.1109/SP.2016.37.

[25] Eman Salem Alashwali and Kasper Rasmussen. 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 2018, Singapore, Singapore, August 8-10, 2018, Proceedings, Part II*. Springer. 2018, pp. 468–487.

[26] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1461–1480. ISBN: 9781450370899. DOI: 10.1145/3372297.3423350. https://kemtls.org/publication/kemtls/.

[27] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella-Béguelin. Proving the tls handshake secure (as it is). In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34*. Springer. 2014, pp. 235–255.

[28] Christian Paquin, Douglas Stebila, and Goutam Tamvada. Benchmarking post-quantum cryptography in tls. In *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*. Springer. 2020, pp. 72–91.

[29] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Post-quantum authentication in tls 1.3: A performance study. *Cryptology ePrint Archive* (2020).

[30] Ruben Gonzalez and Thom Wiggers. Kemtls vs. post-quantum tls: Performance on embedded systems. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2022, pp. 99–117.

[31] Trevor Perrin. *The noise protocol framework, 2016*. http://noiseprotocol.org/noise.pdf.

[32] Neal Koblitz, Alfred Menezes, and Scott Vanstone. The state of elliptic curve cryptography. *Designs, codes and cryptography* 19 (2000), pp. 173–193.

[33] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. *Open Whisper Systems* 283 (2016), p. 10.

[34] Noson S Yanofsky and Mirco A Mannucci. *Quantum computing for computer scientists*. Cambridge University Press, 2008.

[35] https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms.

[36] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv* 4244 (1978), pp. 114–116.

[37] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic mceliece (2017).

[38] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 353–367.

[39] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), pp. 238–268.

[40] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon. *Post-Quantum Cryptography Project of NIST* (2020).

[41] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The sphincs+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2019, pp. 2129–2146.

[42] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Philip R Zimmermann. Post-quantum wireguard. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 304–321.

[43] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[44] Bruno Blanchet. Symbolic and computational mechanized verification of the arinc823 avionic protocols. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. 2017, pp. 68–82. DOI: 10.1109/CSF.2017.7.

[45] Nils Mäurer, Thomas Gräupl, Corinna Schmitt, Gabi Dreo Rodosek, and Helmut Reiser. Advancing the security of ldacs. *IEEE Transactions on Network and Service Management* 19.4 (2022), pp. 5237–5251. DOI: 10.1109/TNSM.2022.3189736.

[46] Deutsches Zentrum fuer Luft- und Raumfahrt e. V. *LDACS implementation and validation*. 2023. https://www.ldacs.com/ldacs1-development/ldacs-implementation-and-validation/ (visited on 07/26/2023).

[47] Thomas Boegl, Mathias Rautenberg, Bernhard Haindl, Christoph Rihacek, Josef Meser, Pierluigi Fantappie, Noppadol Pringvanich, John Micallef, Klauspeter Hauf, John MacBride, Philippe Sacre, Bart van den Einden, Thomas Gräupl, and Michael Schnell. *LDACS white paper – a roll-out scenario*. WORKING PAPER. ICAO, Oct. 2019.

[48] Wouter Castryck and Thomas Decru. *An efficient key recovery attack on sidh*. 2022. https://eprint.iacr.org/2022/975.

[49] EUROCONTROL. *Aeronautical mobile airport communications system datalink*. 2023. https://www.eurocontrol.int/system/aeronautical-mobile-airport-communications-system-datalink (visited on 07/27/2023).

[50] WiMAX Forum. *Aeromacs*. 2023. https://wimaxforum.org/Page/AeroMACS (visited on 07/27/2023).

[51] WiMAX Forum. *Wimax forum security*. 2023. https://wimaxforum.org/Page/Security (visited on 07/27/2023).

[52] The Open Quantum Safe Project. *X.509 | open quantum safe*. 2023. https://openquantumsafe.org/applications/x509.html (visited on 09/11/2023).

[53] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions (2015).

[54] Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *Annual Cryptology Conference*. Springer. 2010, pp. 631–648.

[55] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karolin Varner, and Bas Westerbaan. *X-wing: The hybrid kem you've been looking for*. Cryptology ePrint Archive, Paper 2024/039. https://eprint.iacr.org/2024/039. 2024. https://eprint.iacr.org/2024/039.

[56] Federico Giacon, Felix Heuer, and Bertram Poettering. Kem combiners. In *Public-Key Cryptography – PKC 2018*. Ed. by Michel Abdalla and Ricardo Dahab. Cham: Springer International Publishing, 2018, pp. 190–218. ISBN: 978-3-319-76578-5.

[57] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. *Summer course "Cryptography and computer security" at MIT* 1999 (1996), p. 1999.

[58] Andrew Morgan and Rafael Pass. On the security loss of unique signatures. In *Theory of Cryptography Conference*. Springer. 2018, pp. 507–536.

[59] Mihir Bellare, Hannah Davis, and Felix Günther. Separate your domains: Nist pqc kems, oracle cloning and read-only indifferentiability. 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*. Springer. 2020, pp. 3–32.

[60] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. 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, 1998 Proceedings 18*. Springer. 1998, pp. 26–45.

A. PRIMITIVES FOR PROTOCOL CONSTRUCTION

A.1. 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 **N**umber 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*.

A.2. 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().
```

A.3. 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().
```

A.4. KEMs

KEM stands for "Key Encapsulation Method". 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.
```

A.4.1. 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.
```

B. X25519KYBER768

Abstract description of the X25519Kyber768 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:Kyber768 +
↪  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:Kyber768 +
↪  KDF:shake256: authenticated";

fn X25519Kyber768::encap(sk_mine: Optional<X25519Kyber768::SecretKey>, pk_theirs:
↪  X25519Kyber768::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) = Kyber768::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 X25519Kyber768::decap(sk_mine: X25519Kyber768::SecretKey, pk_theirs:
↪  Optional<X25519Kyber768::PublicKey>, ct: X25519Kyber768::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 = Kyber768::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;
}
```

C. X25519KYBER768DILITHIUM

Abstract description of the X25519Kyber768Dilithium 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:Kyber768 +
↪  Sig:Dilithium3 + 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:Kyber768 +
↪  Sig:Dilithium3 + KDF:shake256: authenticated";

fn X25519Kyber768Dilithium::encap(sk_mine: Optional<X25519Kyber768Dilithium::SecretKey>,
↪  pk_theirs: X25519Kyber768Dilithium::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) = Kyber768::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; Dilithium3::SIGNATURE_LENGTH]
  } else {
    Dilithium3::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 + Dilithium3::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 X25519Kyber768::decap(sk_mine: X25519Kyber768::SecretKey, pk_theirs:
↪  Optional<X25519Kyber768::PublicKey>, ct: X25519Kyber768::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 = Kyber768::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 + Dilithium3::SIGNATURE_LENGTH];

  // If a signature key is given, validate the signature
  if skr3 != None {
    Dilithium3::verify(pks3, sig, kc)
```

```
  }

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

  return ko;
}
```