Hybrid Post-Quantum + Classical Secure Boot for SmallSat Avionics: ML-DSA/ECC Firmware Authentication with Fail-Safe Updates

Daniele Lacamera
wolfSSL Inc.
daniele@wolfssl.com

***Abstract*—Satellites are often expected to operate unattended for years while still receiving security and safety updates. Long mission lifetimes also increase exposure to future cryptanalytic advances, including large-scale quantum attacks against today's signature schemes. We present a hybrid secure-boot and firmware-authentication design aligned with IETF RFC 9019, where the bootloader acts as the firmware verifier. A signed manifest binds the image digest, target identity, and version metadata, and the boot decision enforces a dual-signature policy: one classical (ECC/ECDSA) signature and one post-quantum (ML-DSA, NIST FIPS 204) signature over the same manifest. The verifier avoids dynamic memory allocation and minimizes pre-boot dependencies to improve predictability and limit the remote attack surface.**

We implement this policy in *wolfBoot*, a portable, publicly available, open-source (GPLv3) secure bootloader.

To address reliability constraints, hybrid verification is paired with a fail-safe update strategy using redundant flash slots. Updates are installed through copy/swap operations with explicit progress markers to tolerate resets or power interruptions. After the first boot of a newly installed image, the application must explicitly confirm operational readiness; if confirmation is missed (e.g., watchdog resets or failed communications bring-up), the boot process automatically reverts to the last known-good image, providing an emergency fallback path.

We report code size, RAM usage, and boot-time latency measurements for hybrid verification on representative embedded targets, and derive configuration guidance for balancing security level, signature size, and time-to-boot constraints in SmallSat systems.

***Index Terms*—SmallSat, secure boot, firmware update, post-quantum cryptography, ML-DSA, ECDSA, DO-178C, wolfBoot.**

I. Introduction

Small spacecraft have moved from short-lived demonstrations toward operational assets that may remain unattended for years. NASA's Small Spacecraft Technology State-of-the-Art reporting describes the growth of SmallSat flight heritage and the increasing role of small spacecraft in commercial, government, private, and academic missions [1], [2]. Those same trends make firmware update and recovery architecture part of mission assurance. Flight software may need fixes for command handling, communications, autonomy, cryptographic libraries, or newly discovered vulnerabilities. At the same time, a remote firmware update is a controlled form of remote code execution. If an attacker can replace flight software, replay an old image, or exploit the update parser, the update mechanism becomes an attack surface rather than a resilience feature.

For a satellite, the failure mode is not only an unauthorized image. An update can be interrupted by a reset, brownout, watchdog event, or single-event effect. A new image can also pass cryptographic checks and still fail to initialize radios, attitude control, or payload interfaces. The practical design objective is therefore to avoid an "orbiting brick": a spacecraft that is intact but no longer commandable because a valid-but-bad firmware image was committed without a reliable way back.

This paper describes how *wolfBoot*-based secure boot can be integrated into a SmallSat avionics firmware-update architecture. The integration uses immutable trust anchors, redundant image slots, copy/swap progress markers, anti-rollback metadata, explicit application confirmation, and boot telemetry. The implementation is also selected to support DO-178C certification planning: it can be configured without dynamic allocation, with compile-time algorithm selection, a small manifest parser, deterministic state transitions, and an operating-system-independent handoff. The contribution is a compact reference architecture that combines existing *wolfBoot* mechanisms into a mission-oriented secure-update policy. It prioritizes three properties.

- *wolfBoot* verifies the first executable image by checking a signed manifest and digest before transferring control.
- The trust policy uses hybrid authentication: both a classical ECDSA signature and a post-quantum ML-DSA signature must verify over the same manifest.
- Update installation is fail-safe: the previous image is preserved until the new image boots and confirms operational readiness. The measurements in this paper show that this policy is practical on modern microcontroller-class avionics.

TABLE I
REPRESENTATIVE UPDATE THREATS AND BOOTLOADER CONTROLS.

Threat or failure	Mission effect	Bootloader control
Malicious image	Unauthorized flight software executes	Signed manifest, trust anchors, key permissions
Corrupt flash data	Undefined behavior or failed boot	Image digest bound to manifest
Replay of old image	Known vulnerability restored	Anti-rollback version policy
Interrupted swap	Partial installation	Redundant slots and progress markers
Bad-but-valid image	Spacecraft becomes uncommandable	Confirmation required; automatic revert
Break in ECDSA assumptions	Classical forgery risk	ML-DSA signature also required
PQC issue or implementation fault	PQC bypass risk	ECDSA signature also required

II. REQUIREMENTS AND THREAT MODEL

RFC 9019 frames the bootloader as the firmware verifier and allows the verifier to use the same manifest metadata that accompanies the update distribution [3]. For SmallSats, that verifier must address both adversarial and non-adversarial failures. The essential requirements are summarized below and mapped to representative controls in Table I.

The bootloader must verify every executable image before use, because hash-only integrity is insufficient when an attacker can replace both image and hash. The security decision should be bound to a manifest containing the image digest, size, version, target identity, algorithm identifiers, and key identifiers. A monotonic version or equivalent replay-resistant policy is required because old firmware can remain validly signed while containing known vulnerabilities. The updater must also survive interrupted copy/swap processing; RFC 9019 explicitly requires that devices not fail when update processing is disrupted by power loss or network interruption [3]. For spacecraft, the same principle extends to resets, watchdogs, contact loss, and radiation-induced faults.

Finally, the boot chain should bound the trusted computing base. Network protocols, file systems, compression formats, dynamic allocators, and complex codecs should remain outside *wolfBoot* whenever they are not required for the boot decision. This reduces the amount of software that must be trusted before the first authenticated image begins executing.

III. WOLFBOOT ARCHITECTURE FOR FAIL-SAFE UPDATES

The architecture places *wolfBoot* in the earliest software execution environment. After reset, the bootloader initializes only the hardware needed to read flash, parse the manifest, verify the image, evaluate rollback state, and transfer control. The application or communications subsystem is responsible for receiving update packages and placing them into the update slot; *wolfBoot* is responsible for deciding whether any candidate image may execute.

A practical SmallSat flash map contains at least four regions: an immutable or tightly controlled bootloader region, an active image slot, an update slot, and a small state area for version counters and copy/swap markers. The manifest is part of the signed update package. It binds the image hash, version, size, key identifiers, and signature objects. For hybrid authentication, the manifest contains two independent signature records over the same firmware metadata: one ML-DSA signature and one ECDSA signature. The public keys are provisioned in a *wolfBoot* keystore, preferably through immutable or manufacturing-controlled storage such as OTP, a secure element, TPM-backed nonvolatile storage, or a controlled HSM-based provisioning process.

The update state machine is as important as the signature check. A candidate image staged in the update slot is not allowed to overwrite the only known-good image irreversibly. Instead, *wolfBoot* installs the update using copy/swap operations with explicit progress markers. If power is lost or a reset occurs during an erase, copy, or verification step, the next boot resumes or rolls back from a well-defined state. When the candidate image first boots, it remains provisional. The application must run its self-tests, bring up the communication path and mission services required by policy, and explicitly confirm the image. If the application fails to confirm, or if a watchdog reset occurs before confirmation, *wolfBoot* reverts to the previous image.

This mechanism separates authenticity from operational readiness. A signature proves that an authorized authority produced the image and that the image has not been modified. It does not prove that the image can talk to the radio, manage power, or support ground-command recovery. Application confirmation closes that gap by making the commit decision depend on demonstrated behavior after first boot. The same state should be exposed to ground telemetry: active version, candidate version, rollback cause, slot state, and signature or manifest error codes are often enough to help operators decide whether to retry, uplink a corrected package, or command safe mode.

The boot flow is intentionally linear.

- Identify the active slot and any pending update slot.
- Parse only the fixed manifest structures required by the selected configuration.
- Compute the image digest and check that the manifest binds the expected size and version.
- Verify the post-quantum signature and the classical signature using keys authorized for that partition.
- Evaluate anti-rollback and update-state rules.

Only after those checks succeed does it update slot metadata and jump to the application entry point. If any decision fails, control remains in the boot policy: either the previous image is restored, the device stays in a safe boot state, or a target-specific recovery path is invoked.

A SmallSat implementation should treat the update slot as untrusted storage until the full manifest and both signatures have been checked. This is particularly important when the update is delivered over a low-rate radio link or stored in external flash. The transport layer may fragment, retransmit, encrypt, or compress the package, but none of those operations should be part of the root of trust. The bootloader sees only

TABLE II
FAIL-SAFE UPDATE STATE CARRIED ACROSS RESETS.

State item	Purpose in a reset-tolerant update
Active version	Identifies the image that is currently allowed to run.
Candidate version	Records the image under installation or first-boot test.
Copy/swap marker	Indicates the exact erase, write, copy, verify, or commit phase reached before reset.
Rollback counter	Prevents replay to stale but still signed firmware.
Confirmation flag	Distinguishes a candidate that merely booted from one that passed mission self-test.
Failure code	Provides telemetry for manifest, digest, signature, version, or confirmation failure.

the final candidate image and its manifest. This separation is useful both for security and for certification: transport complexity can evolve in the application without changing the boot decision logic.

IV. Manifest, Keystore, and Signing Flow

The manifest is the bridge between the ground signing process and the on-board verifier. It should bind the firmware bytes to a target identity, version, size, hash algorithm, signing algorithms, key identifiers, and any partition or component permissions. Binding this metadata prevents ambiguous update packages: an image signed for a payload controller should not be accepted by an avionics controller, and an image signed for a newer software baseline should not be replayed as an emergency downgrade unless mission policy explicitly permits it.

The keystore is the corresponding on-device authorization database. In a hybrid design, the keystore should not treat the post-quantum and classical keys as interchangeable labels on the same authority. Each key has an algorithm type, a slot, and a permission mask. A flight configuration can therefore require a valid ML-DSA signature from one authority and a valid ECDSA signature from another authority, both scoped to the target partition. This provides defense in depth against algorithmic breaks, implementation faults, and operational key compromise. If one signing path is suspected, the mission can rotate or revoke it without relaxing the other path.

Manufacturing should provision trust anchors through a process that is at least as controlled as the rest of the avionics build. One-time-programmable memory is attractive when available because it prevents accidental in-field modification. A secure element, TPM-backed nonvolatile storage, or a manufacturing HSM process can provide similar process-level assurance. The essential property is that an attacker who can write ordinary application flash cannot also rewrite the public keys or downgrade the verification policy.

The signing pipeline can be split so that private keys never reside on the build host. A reproducible build produces an unsigned image and manifest digest. The digest is submitted to the ML-DSA signing authority and to the ECDSA signing authority, which may be separate HSM-backed services. The two signatures are returned and assembled into the update package. On the spacecraft, *wolfBoot* recomputes the digest, validates both signatures, checks the version policy, and only then considers the image eligible for execution. This split flow also supports audits: the mission can record which authority signed which version, which key slot was used, and which spacecraft or constellation subset was authorized to receive the package.

V. Hybrid Signature Policy

Long SmallSat mission lifetimes create cryptographic transition risk. A spacecraft integrated today may still be operational after large-scale quantum computers or other cryptanalytic advances change the assurance level of classical public-key algorithms. NIST FIPS 204 standardizes ML-DSA and states that it is designed for security against adversaries with large-scale quantum computers [6]. A sudden switch from classical signatures to a single post-quantum algorithm, however, also creates migration risk. Implementations and operational processes for new algorithms need time to mature.

The proposed policy is therefore AND-composed hybrid authentication. *wolfBoot* accepts a flight image only when both verifications succeed: an ML-DSA signature and an ECDSA signature over the same manifest. This is stricter than a migration policy in which either signature is enough. An OR-composed policy can help during staged deployment, but it reduces security to the weaker accepted path. For flight images, the AND policy forces an attacker to defeat two independent assumptions or compromise two signing authorities.

Two profiles are useful in practice. A balanced profile uses ML-DSA-65 with ECDSA P-384, providing a post-quantum signature and a widely deployed high-assurance classical curve with measured sub-500 ms boot latency in the benchmark target described below. A high-assurance profile uses ML-DSA-87 with ECDSA P-521, increasing the security margin at the cost of larger metadata and longer classical verification time. Both profiles are implemented as *wolfBoot* configuration choices rather than changes to the application.

Key management should preserve the independence that the hybrid design assumes. The ML-DSA and ECDSA keys should have separate key identifiers, separate rotation plans, and preferably separate HSM-backed signing workflows. Permission masks should restrict which partitions a key may sign; a payload update key should not be able to sign flight-critical avionics unless mission policy explicitly allows it. The manifest format and keystore should also reserve space for future algorithms so that crypto agility does not require a flash-map redesign.

VI. DO-178C-Certifiable Configuration

wolfBoot influences safety because it selects which software is allowed to execute. For avionics projects requiring DO-178C evidence [11], the flight configuration should be treated as a bounded software item rather than as a general-purpose updater. The design choices are familiar certification aids: static memory use, no dynamic allocation, a small parser, deterministic branch behavior, limited algorithm set, OS independence, and traceable reject/rollback requirements.

TABLE III
EXAMPLE EVIDENCE MAPPING FOR A CERTIFIABLE WOLFBOOT CONFIGURATION.

Bootloader property	Certification-relevant evidence
Static memory use	Linker map, stack analysis, absence of allocation APIs.
Manifest parser	Low-level requirements for every accepted and rejected field.
Hybrid verification	Tests for missing, wrong-key, corrupt, and mismatched signatures.
Update state machine	Reset-injection tests at each progress marker.
Rollback policy	Version-counter tests for stale, equal, and forward versions.
Handoff path	Interface tests proving only verified images are entered.

The relevant certification argument is not that a generic feature-rich bootloader is automatically certifiable. It is that a constrained *wolfBoot* configuration can be kept small enough for rigorous requirements, code review, structural coverage, robustness testing, and tool-chain control. The trusted code path consists of startup, flash access, manifest parsing, hash verification, signature verification, slot selection, anti-rollback checks, update-state recovery, and handoff. Each boot decision can be linked to a high-level requirement and a low-level design element.

Robustness testing should include malformed manifests, missing signatures, wrong key identifiers, stale versions, corrupt image data, interrupted swap operations, reset at each progress marker, invalid confirmation state, and corrupted state metadata. Worst-case execution-time analysis should include hashing the maximum supported image, verifying both signatures, and completing the longest permitted copy/swap recovery path. *wolfSSL* has published DO-178C DAL A support material for *wolfCrypt* and describes secure boot, secure firmware update, and avionics cybersecurity use cases for the *wolfSSL* ecosystem [8]–[10]. The certification status of a final spacecraft system still depends on the exact target, configuration, tool chain, verification evidence, and certification authority; the architectural point is to keep the boot component sufficiently deterministic and bounded to support that evidence.

VII. IMPLEMENTATION AND MEASUREMENTS

A. Reference Configuration

wolfBoot is a portable, OS-independent secure bootloader built on *wolfCrypt*. The public *wolfBoot* materials describe a minimal HAL, firmware authentication and update mechanisms, anti-rollback support, restoration of a previous image when a new image is not confirmed, and support for SHA-2/SHA-3, ECDSA/RSA, ML-DSA, LMS, XMSS, and hybrid authentication [4], [5], [7]. Representative build-time settings are shown in Table IV. The important feature is that the signature policy is compiled into the bootloader image rather than negotiated by the candidate application.

The signing pipeline can be local for development, but mission builds should keep private keys outside the build host when assurance requires separation of duties. A practical flow computes the manifest digest, obtains the ML-DSA and ECDSA signatures from separate HSM-backed authorities, and assembles the package with both signature records.

TABLE IV
REPRESENTATIVE HYBRID *wolfBoot* BUILD SETTINGS.

`Setting`	Balanced	High assurance
`SIGN`	ML_DSA	ML_DSA
`ML_DSA_LEVEL`	3	5
`SIGN_SECONDARY`	ECC384	ECC521
`HASH`	SHA3	SHA3
`IMAGE_HEADER_SIZE`	8192	12288

TABLE V
SELECTED *wolfBoot* MEASUREMENTS FOR A ROUGHLY 100 KB IMAGE ON A MODERN CORTEX-M33-CLASS MICROCONTROLLER. LATENCY IS NORMALIZED TO A 250 MHZ OPERATING POINT.

Configuration	Policy	Code B	Stack B	Header B	Lat. ms
SHA-2 baseline	Integrity	10600	1216	1024	71
ML-DSA-65	PQC	22164	25000	8192	127
ECDSA P-384	Classical	25656	11216	1024	380
ML-DSA-65 + P-384	Hybrid	38820	25000	8192	437
ML-DSA-87	PQC	22804	25000	12288	164
ECDSA P-521	Classical	29292	8480	1024	517
ML-DSA-87 + P-521	Hybrid	43700	25000	12288	611

B. Benchmark Method

The benchmark objective was to quantify the cost of adding post-quantum and hybrid authentication to *wolfBoot* on a microcontroller-class target. The source measurements were collected on physical hardware using a GPIO timing method: reset is released, *wolfBoot* verifies the candidate image and transfers control, and the application toggles a GPIO to mark boot completion. The application image is approximately 100 kB. To keep the paper vendor-neutral, Table V reports the target as a modern Cortex-M33-class microcontroller and normalizes latencies to a 250 MHz operating point. Actual values will vary with flash wait states, caches, compiler flags, memory topology, and hardware acceleration; certification projects should repeat measurements on the flight configuration.

The results in Table V and Fig. 1 show that ML-DSA verification is not the dominant time cost in this implementation. Standalone ML-DSA-65 is approximately 127 ms, while ECDSA P-384 is approximately 380 ms. Standalone ML-DSA-87 is approximately 164 ms, while ECDSA P-521 is approximately 517 ms. The balanced hybrid profile boots in approximately 437 ms, and the high-assurance hybrid profile boots in approximately 611 ms. The hybrid cost is lower than the naive sum of standalone measurements because manifest processing and image hashing are shared.

The main resource penalty of ML-DSA is metadata size and working memory. The balanced profile requires an 8 kB image header; the high-assurance profile requires 12 kB. The measured stack budget for ML-DSA configurations is 25 kB. These values are practical on many modern microcontrollers,

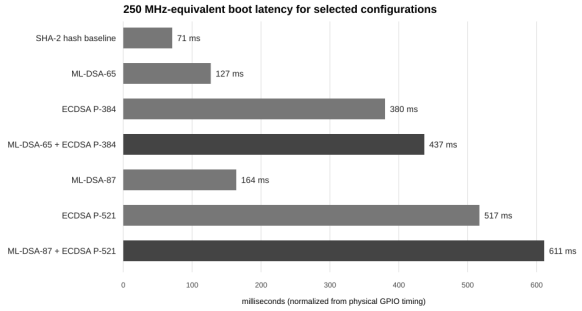

Fig. 1. Boot latency for selected classical, post-quantum, and hybrid configurations, normalized to a 250 MHz Cortex-M33-class target.

but they must be included in flash partition and RAM planning from the start.

VIII. Configuration Guidance and Lifecycle

Configuration should be selected by mission lifetime, update frequency, attack model, and boot-time budget. Short educational missions may deploy classical ECDSA initially while reserving header space and key slots for post-quantum migration. Operational low-Earth-orbit missions with multi-year lifetimes are a good fit for ML-DSA-65 plus ECDSA P-384 with both signatures required. High-value or long-lived missions can select ML-DSA-87 plus ECDSA P-521 when the extra header space and latency are acceptable. Strict watchdog platforms should measure the balanced profile on target hardware and optimize the classical verification path, because the measured hybrid profiles are dominated by ECDSA verification rather than by ML-DSA verification.

Secure boot is not the whole spacecraft security architecture. It authenticates the first image but does not prove the application has no defects or that the running system cannot be exploited after boot. A complete mission design should combine secure boot with command authentication, secure communications, watchdog and safe-mode design, memory protection, least-privilege partitioning, and fault-management logic. Measured boot can complement secure boot by producing evidence of which image executed, while target-specific worst-case execution-time and fault-injection campaigns can turn the design into certification evidence. In the proposed *wolfBoot* architecture, the *wolfCrypt* configuration can also be aligned with a wider *wolfSSL* integration for TLS or DTLS communication security, avoiding separate cryptographic stacks where platform constraints permit it.

IX. Operational Deployment Considerations

A secure boot architecture becomes valuable only when it is embedded in an operational update process. For a single spacecraft, the process should define when update packages may be accepted, what telemetry is required before installation, how long a candidate is allowed to remain unconfirmed, and what ground action follows an automatic revert. For a constellation, the same policy should also support staged rollout. A new image can be signed once but authorized operationally for a subset of spacecraft, reducing the consequence of an unexpected platform interaction. The manifest and ground database should make it clear which spacecraft, processor, partition, and mission mode each package targets.

Boot telemetry should be small but explicit. At minimum, housekeeping should report active version, previous version, candidate version, confirmation state, rollback cause, and the last verification failure class. It is usually not necessary to downlink the entire manifest or signature result; a compact numeric code can identify whether the rejection came from malformed metadata, digest mismatch, missing ML-DSA signature, missing ECDSA signature, wrong key identifier, stale version, failed copy/swap recovery, or confirmation timeout. These fields help operators distinguish a communications problem from a boot-policy problem during a short contact window.

The confirmation step should be tied to mission-specific readiness rather than a trivial application call. A communications satellite may require radio bring-up, command authentication, nonvolatile health-state access, and a watchdog margin before confirming. An Earth-observation payload may require payload-controller enumeration and thermal checks. A safe-mode image may confirm after a smaller set of tests if its purpose is command recovery rather than full mission service. The policy should be conservative: if the new image cannot prove it can receive future commands, it should not permanently replace the last known-good image.

The ground signing process should also include emergency procedures. It is tempting to keep a bypass key or a debug boot path for late mission recovery, but such paths can undermine the entire secure-boot argument. A safer approach is to pre-provision a minimal recovery image signed under the same hybrid policy and to keep its versioning compatible with anti-rollback rules. If a mission requires a break-glass capability, it should be documented as a separate hazard with access controls, dual authorization, logging, and a test plan. The goal is to make recovery possible without creating an undocumented alternative root of trust.

X. Limits and Future work

The architecture deliberately narrows the problem to boot-time authenticity, update recovery, and cryptographic transition. Software alone cannot replace system-level fault management. Radiation can still corrupt RAM or peripherals after boot. A signed application can still contain a functional defect. A compromised ground system can still request an authorized but operationally dangerous action. The bootloader should therefore be one layer in a mission security and safety case that also covers command authentication, encrypted communications, watchdog policy, safe-mode entry, memory protection, and ground procedures. Related *wolfSSL* components can support several of those functions, but they are outside the scope of this paper. The secure boot mechanism mainly ensures that those components can themselves be updated under a controlled authentication and recovery policy.

Future work includes target-specific fault-injection campaigns, measured-boot or remote-attestation extensions, secure-enclave or secure-element key storage, and integration with hardware cryptographic engines. The compact architecture proposed in this paper should be considered early in mission design, when flash layout, watchdog policy, provisioning, and update operations can still be shaped around a secure and updateable flight baseline.

XI. Conclusion

SmallSat firmware updates must be secure and recoverable. Authentication without rollback can leave a satellite running valid but operationally broken firmware, while rollback without authentication can become a remote code-execution mechanism. A *wolfBoot*-based architecture can combine both properties: hybrid ML-DSA/ECDSA authentication for cryptographic transition readiness, and fail-safe redundant-slot updates with application confirmation to avoid committing an uncommandable image.

The measured results indicate that hybrid authentication is practical on a modern Cortex-M33-class microcontroller. A balanced ML-DSA-65/ECDSA P-384 profile boots in approximately 437 ms at a 250 MHz-equivalent operating point, and a high-assurance ML-DSA-87/ECDSA P-521 profile boots in approximately 611 ms. With a small, statically configured, OS-independent boot path, the same design can support DO-178C certification planning while remaining cryptographically agile for long-lived spacecraft.

References

[1] NASA Small Spacecraft Systems Virtual Institute, "State-of-the-Art of Small Spacecraft Technology," 2024 Edition, Feb. 2025. [Online]. Available: https://www.nasa.gov/smallsat-institute/sst-soa/

[2] NASA, "Small Spacecraft Technology State of the Art Report: SmallSat Avionics," 2024 Edition, Ch. 8, Feb. 2025. [Online]. Available: https://www.nasa.gov/wp-content/uploads/2025/02/8-soa-small-spacecraft-avionics-2024.pdf

[3] B. Moran, H. Tschofenig, D. Brown, and M. Meriac, "A Firmware Update Architecture for Internet of Things," IETF RFC 9019, Apr. 2021.

[4] wolfSSL, "wolfBoot Secure Bootloader." [Online]. Available: https://www.wolfssl.com/products/wolfBoot/

[5] wolfSSL, "wolfBoot: Secure Boot now with support for FIPS 204 ML-DSA post-quantum signature algorithm," Oct. 2024. [Online]. Available: https://www.wolfssl.com/wolfboot-secure-boot-now-with-support-for-fips-204-ml-dsa-post-quantum-signature-algorithm/

[6] NIST, "Module-Lattice-Based Digital Signature Standard," FIPS 204, Aug. 2024.

[7] wolfSSL, "wolfBoot release: v.2.3.0," Nov. 2024. [Online]. Available: https://www.wolfssl.com/wolfboot-release-v-2-3-0/

[8] wolfSSL, "wolfSSL Support for DO-178C DAL A." [Online]. Available: https://www.wolfssl.com/wolfssl-support-for-do-178c-dal-a/

[9] wolfSSL, "Achieving Avionics Security with DO-178C-Certified Cryptography," Feb. 2026. [Online]. Available: https://www.wolfssl.com/achieving-avionics-security-with-do-178c-certified-cryptography/

[10] wolfSSL, "Cybersecurity for Avionics." [Online]. Available: https://www.wolfssl.com/aerospace-innovations/

[11] RTCA, "Software Considerations in Airborne Systems and Equipment Certification," DO-178C, 2011.