Kilobyte-Bandwidth Subliminal Channels in FIPS 204 ML-DSA via Packed-Commitment Embedding

Mounir IDRASSI[1]

[1]AM Crypto, mounir@amcrypto.jp

January 13, 2026

Abstract

Galteland and Gjøsteen [11] observe that Dilithium-family signatures admit *broadband* subliminal channels in a *secret-key-assisted* setting where the receiver can reconstruct the signer's hidden commitment from a public signature. This note is a standards-specific instantiation of that observation for FIPS 204 ML-DSA [1]. We do *not* claim a new subliminal-channel technique: our goal is to make the FIPS 204 patch point and byte-level embedding interface explicit, and to list the resulting capacities for the approved parameter sets.

Two FIPS 204 facts drive the instantiation. First, for any *accepted* signature, the commitment vector $\mathbf{y}$ is recoverable from $(c, \mathbf{z})$ given $\mathbf{s}_1$ via $\mathbf{y} = \mathbf{z} - c\mathbf{s}_1$. Second, the standardized γ_1 values make the FIPS 204 $\mathsf{BitUnpack}(\cdot, \gamma_1 - 1, \gamma_1)$ mapping have *full support* on its fixed-length byte input, so the packed commitment is a bijection to a fixed-length byte string. We embed an L-byte payload by XOR-masking a pseudorandom packed container and decoding it with the standard $\mathsf{BitUnpack}$ routine: the resulting signatures verify under unmodified verifiers.

We implemented the patch in the `mldsa-native` C library and validated round-trip extraction, abort-rate statistics, and distribution sanity checks for all three parameter sets. The forked repository is available at [3]. Selected plots are summarized in the main text, with a full plot inventory in the appendix.

The per-signature covert capacity is $32\ell w - 32$ bytes, where $w = \log_2(2\gamma_1)$, namely 2,272 bytes for ML-DSA-44, 3,168 bytes for ML-DSA-65, and 4,448 bytes for ML-DSA-87. As in [11], extraction requires $\mathbf{s}_1$ (sufficient, together with public data, for existential forgery via known alternative signing procedures), so the relevant setting is kleptographic/ASA-style secret sharing rather than public tagging.

1 Introduction and scope

Subliminal channels (Simmons) allow a malicious signer to communicate covertly by steering per-signature randomness while preserving public verifiability [6, 7]. They are a foundational primitive for kleptography and algorithm-substitution attacks (ASAs) [13, 14, 15, 16], and have been demonstrated concretely in high-speed signature settings (e.g., EdDSA) [9, 10]. Early subliminal-free countermeasures exist in other models [8], and more recent mitigations include reverse-firewall designs [17, 18].

Galteland and Gjøsteen [11] analyze *secret-key-assisted* broadband channels in post-quantum signature candidates, including Dilithium-family designs. This note is intentionally narrow and incremental over [11]: it specializes the GG19 construction to the finalized FIPS 204 ML-DSA specification and spells out an *implementer-facing, byte-accurate* embedding route that uses only standardized packing/unpacking routines. Concretely, we provide:

Table 1: Notation summary.

Symbol	Meaning
ML-DSA	FIPS 204 ML-DSA signature scheme.
ℓ	ML-DSA parameter: length of commitment vector $\mathbf{y}$.
γ_1	Commitment bound parameter in ML-DSA.
w	Bits per coefficient, $w = \log_2(2\gamma_1)$.
N	Packed commitment size, $N = 32\ell w$ bytes.
L	Payload length, $L \leq N - 32$ bytes.
$\mathbf{y}$	Commitment vector sampled in signing.
$\mathbf{s}_1$	Secret key vector used for extraction.
μ	Message representative in FIPS 204 signing.
$\tilde{c}$, c	Packed challenge bytes and derived challenge polynomial.
r	32-byte per-attempt seed in the packed container.
K_{sc}	Subliminal-channel key.
$\mathsf{XOF}_{K_{\mathsf{sc}}}$	Keyed XOF/PRF used to mask the payload.
"MLDSA-SC"	Domain-separation string for $\mathsf{XOF}_{K_{\mathsf{sc}}}$.
κ	Retry counter in the abort loop.

- an explicit packed-domain embedding interface for the FIPS 204 commitment sampler,
- exact per-signature capacities for ML-DSA-44/65/87,
- a reference implementation patch in a C library, and
- an experimental validation summary with selected plots (and a full appendix inventory).

We assume the default randomized (hedged) signing mode of FIPS 204: deterministic signing removes the signer-controlled randomness surface, embedding would require deviating from the standardized deterministic procedure, so we exclude it.

2 Two FIPS 204 properties enabling packed-domain embedding

We recall only what is needed from FIPS 204 [1]. We work in the standard ring $\mathcal{R} = \mathbb{Z}_q[X]/(X^{256}+1)$ and write $\mathcal{R}^\ell$ for length-ℓ vectors. In ML-DSA.Sign_internal, the signer computes a 64-byte message representative $\mu \leftarrow H(\mathsf{tr} \parallel M')$ (with M' the formatted message input as defined by the standard), and then runs a Fiat–Shamir-with-aborts loop. In each loop iteration, the signer samples a commitment vector $\mathbf{y} \in \mathcal{R}^\ell$ with coefficients in $[-\gamma_1 + 1, \gamma_1]$ using ExpandMask, derives a sparse challenge polynomial $c \in \mathcal{R}$ from the commitment and μ, and computes the response $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$, retrying if the standard abort predicate fails.

2.1 Recovering y from an accepted signature (secret-key-assisted)

Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ be an *accepted* signature, where $\tilde{c}$ denotes the packed challenge bytes: let $c = \mathsf{SampleInBall}(\tilde{c})$. Rearranging the signing equation gives

$$\mathbf{y} = \mathbf{z} - c\mathbf{s}_1.$$

In accepted signatures, FIPS 204 enforces $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and parameters imply $\|c\mathbf{s}_1\|_\infty \leq \beta$, so the recovered $\mathbf{y}$ lies in the intended coefficient range and is unambiguous when interpreted as centered representatives [1]. This is exactly the secret-key-assisted recovery step used in [11].

2.2 Full-support packing for standardized γ_1

FIPS 204 packs and unpacks polynomials using $\mathsf{BitPack}(w, a, b)$ and $\mathsf{BitUnpack}(v, a, b)$ [1]. For the commitment range $[-\gamma_1 + 1, \gamma_1]$, we set $(a, b) = (\gamma_1 - 1, \gamma_1)$ so that $[-a, b] = [-\gamma_1 + 1, \gamma_1]$. The packed bit width per coefficient is

$$w = \mathsf{bitlen}(a + b) = \mathsf{bitlen}(2\gamma_1 - 1) = \log_2(2\gamma_1),$$

where the last equality holds for the standardized parameter sets because γ_1 is a power of two.

Crucially, FIPS 204 states that $\mathsf{BitUnpack}(\cdot, a, b)$ has *full support* on its fixed-length byte input when $a+b+1$ is a power of two [1]. Here $a+b+1 = 2\gamma_1$, which is a power of two for all standardized parameter sets. Therefore, *every* w-bit chunk maps to a valid coefficient in $[-\gamma_1 + 1, \gamma_1]$, and the mapping between a length-$32w$-byte string and a 256-coefficient polynomial in that range is a bijection. Componentwise for $\mathbf{y} \in \mathcal{R}^\ell$, the packed representation has fixed length

$$N = \frac{256\ell w}{8} = 32\ell w \quad \text{bytes.}$$

3 Construction: embedding in the packed commitment sampler

3.1 Keys, interface, and notation

Let K_{sc} be a secret key used only for the subliminal channel. We instantiate $\mathsf{XOF}_{K_{\mathsf{sc}}}$ as a keyed PRF/stream generator using SHAKE256 with a secret-prefix absorption of "MLDSA-SC" $\|$ K_{sc} $\|$ μ $\|$ r: standardized keyed variants such as KMAC256 or keyed cSHAKE256 with the same domain string are drop-in replacements. We use a public domain separator "MLDSA-SC" for all invocations.

We write $\mathsf{XOF}_{K_{\mathsf{sc}}}(m, n)$ for the n-byte output of the keyed XOF on input string m. Inputs are encoded as a fixed-length concatenation "MLDSA-SC" $\|$ K_{sc} $\|$ μ $\|$ r: since μ is 64 bytes and r is 32 bytes, this encoding is unambiguous. If a variable-length μ is used in other settings, the concatenation should be length-prefixed. We define the external interface:

$$\mathsf{SC.Sign}(sk, M, W) \to \sigma, \qquad \mathsf{SC.Extract}(sk, M, \sigma, L) \to W' \text{ (or } \perp \text{ if optional validation is used).}$$

Here W is an L-byte payload with $L \le N - 32$, and extraction requires secret material containing $\mathbf{s}_1$ and K_{sc}.

3.2 Embedding: sampling y from a masked packed container

We reserve the first 32 bytes of the packed container for a per-attempt seed r and embed the payload into the remaining $N - 32$ bytes. The extractor will recover r by repacking $\mathbf{y}$.

3.3 Signing and retry behavior

SC.Sign runs the standard FIPS 204 signing algorithm unchanged *except* that the commitment sampler $\mathbf{y} \leftarrow \mathsf{ExpandMask}(\cdot)$ is replaced by Algorithm 1, with W fixed as an additional input.

Retry behavior. The FIPS 204 abort/retry loop is preserved. If an attempt aborts, the signer increments the retry counter and repeats the entire attempt with the *same* μ and payload W, but with a *fresh* r (hence a fresh $\mathbf{y}$) because SC.EmbedY is invoked again. No payload-dependent acceptance predicate is introduced, and no additional rejection sampling loop is added beyond the standard abort predicate.

Algorithm 1 $\mathsf{SC.EmbedY}(K_{\mathsf{sc}}, \mu, W)$ producing $\mathbf{y} \in \mathcal{R}^{\ell}$

Require: Channel key K_{sc}, message representative μ (as in FIPS 204), payload W as L bytes with $L \leq N - 32$, where $N = 32\ell w$

Ensure: Commitment $\mathbf{y} \in \mathcal{R}^{\ell}$ with coefficients in $[-\gamma_1 + 1, \gamma_1]$

Sample $r \xleftarrow{\$} \{0,1\}^{256}$ ▷ r is 32 bytes
$S \leftarrow \mathsf{XOF}_{K_{\mathsf{sc}}}(\text{“MLDSA-SC”} \parallel K_{\mathsf{sc}} \parallel \mu \parallel r,\ N - 32)$ ▷ S is $(N - 32)$ bytes
$S[0..L-1] \leftarrow S[0..L-1] \oplus W$
$B \leftarrow r \parallel S$ ▷ $|B| = N$ bytes
Parse B into ℓ blocks $B_0, \ldots, B_{\ell-1}$ of $32w$ bytes each
for $i = 0$ to $\ell - 1$ **do**
    $y_i \leftarrow \mathsf{BitUnpack}(B_i,\ \gamma_1 - 1,\ \gamma_1)$
end for
return $\mathbf{y} = (y_0, \ldots, y_{\ell-1})$

Algorithm 2 $\mathsf{SC.Extract}(sk, M, \sigma, L)$

Require: Secret key material containing $\mathbf{s}_1$ and K_{sc}; message M; signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$; payload length L

Ensure: L-byte string W' (or $\perp$ if validation is enabled)

(Optional) If ML-DSA.Verify rejects (pk, M, σ) then return $\perp$
Compute μ exactly as in FIPS 204 signing
$c \leftarrow \mathsf{SampleInBall}(\tilde{c})$
$\mathbf{y} \leftarrow \mathbf{z} - c\mathbf{s}_1$
If $L > N - 32$, return $\perp$
$B \leftarrow \mathsf{BitPack}(\mathbf{y},\ \gamma_1 - 1,\ \gamma_1)$ ▷ $|B| = N$ bytes
Parse B as $B = r \parallel S'$ where r is the first 32 bytes
$S \leftarrow \mathsf{XOF}_{K_{\mathsf{sc}}}(\text{“MLDSA-SC”} \parallel K_{\mathsf{sc}} \parallel \mu \parallel r,\ N - 32)$
$W' \leftarrow S'[0..L-1] \oplus S[0..L-1]$
return W'

3.4 Extraction and optional validation

Given $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$, the extractor recomputes μ exactly as in FIPS 204 signing, derives $c = \mathsf{SampleInBall}(\tilde{c})$, recovers $\mathbf{y} = \mathbf{z} - c\mathbf{s}_1$, and repacks $\mathbf{y}$ into the byte container B.

Optional detection and centered lift. Without extra structure, SC.Extract outputs an L-byte string on any valid signature: on non-channel signatures this output is typically pseudorandom. To obtain explicit detection, reserve a small header inside W, e.g. $W = \mathsf{magic} \parallel \mathsf{meta} \parallel \mathsf{tag}$ where $\mathsf{tag} = \mathsf{MAC}_{K_{\mathsf{sc}}}(\mu \parallel \mathsf{meta})$, and accept only if the tag verifies. If the tag is t bytes, capacity is reduced by t: for example, a 32-byte tag reduces ML-DSA-44 capacity from 2,272 to 2,240 bytes. Coefficients are interpreted as centered representatives consistent with the acceptance bounds in FIPS 204: implementations must avoid reducing modulo q when forming $\mathbf{y} = \mathbf{z} - c\mathbf{s}_1$.

Threat model (secret-key-assisted). We consider settings where the party performing extraction holds $\mathbf{s}_1$ (as defined in FIPS 204 [1]) and the subliminal-channel key K_{sc}. An external warden sees (pk, M, σ) and adaptive signing queries but does not learn $\mathbf{s}_1$ or K_{sc}. The intended use is covert communication between parties already in possession of signing-equivalent secrets (exfiltration or

coordination), not public tagging. Knowledge of $\mathbf{s}_1$ alone suffices, together with public data, to enable existential forgery via known alternative signing procedures [11, 12].

Practical deployment scenarios. Although extraction is *secret-key-assisted* (it requires $\mathbf{s}_1$ and K_{sc}), the channel is still operationally relevant in deployments where signing-equivalent secrets are replicated, escrowed, or derivable by multiple principals. In such settings, signatures that are *already* generated and transported for legitimate reasons (software-update signing, authentication tokens, signed audit records, certificate issuance, etc.) can double as a high-bandwidth covert channel without introducing additional network flows. Examples include:

- **Key replication across a signing fleet.** Large services often replicate a signing key across multiple nodes or HSMs for availability: a compromised signing node can embed telemetry or exfiltrated secrets into routine signatures, and any other key-holding insider (or compromised node) can extract it.
- **Escrow/derivation models.** In manufacturing or managed-device settings where device signing keys are escrowed or derived from a vendor-held master, the vendor (or a supply-chain adversary with that derivation capability) can decode payloads from field-observed signatures.
- **Air-gapped or tightly monitored signing environments.** If signatures are one of the few permitted outputs (e.g., release-signing in an offline build pipeline), a subverted signer that *also* shares the required extraction secrets can leak kilobytes per signature via artifacts that are expected to be public.

These scenarios differ from public tagging: the primary risk is covert coordination or data exfiltration between parties that already possess (or can derive) signing-equivalent secrets, i.e., an ASA/kleptographic setting with key sharing rather than a public steganographic watermark.

4 Analysis

4.1 Correctness

Claim. If $\sigma \leftarrow \mathsf{SC.Sign}(sk, M, W)$ is an accepted signature embedding an L-byte payload W, then $\mathsf{SC.Extract}(sk, M, \sigma, L)$ outputs W.

Proof sketch. Algorithm 1 builds a packed container

$$B = r \parallel S', \qquad S' = S \oplus (W \parallel 0),$$

with $S = \mathsf{XOF}_{K_{\mathsf{sc}}}(\text{“MLDSA-SC”} \parallel K_{\mathsf{sc}} \parallel \mu \parallel r, \cdot)$. By the full-support/bijectivity of the FIPS 204 packing/unpacking for standardized γ_1, decoding B yields a unique $\mathbf{y}$ and re-encoding $\mathbf{y}$ returns the same B. Given an accepted signature, secret-key-assisted recovery produces the same $\mathbf{y}$ from $(c, \mathbf{z}, \mathbf{s}_1)$, hence the same B, hence the same r and S', and XOR cancellation yields W. This argument assumes the standard packing correctness, fixed-length concatenation for (μ, r), and deterministic behavior of $\mathsf{XOF}_{K_{\mathsf{sc}}}$. Implementers must interpret $\mathbf{z}$ and $c\mathbf{s}_1$ as centered representatives (not modulo q) consistent with the acceptance bounds in FIPS 204, otherwise the recovery $\mathbf{y} = \mathbf{z} - c\mathbf{s}_1$ can fail.

4.2 Bandwidth

With $N = 32\ell w$ bytes and 32 bytes reserved for r, the per-signature capacity is

$$N = \frac{256\ell w}{8} = 32\ell w, \qquad \mathsf{cap}(\mathsf{ML\text{-}DSA}) = N - 32 = 32\ell w - 32 \quad \text{bytes.}$$

Table 2: Subliminal capacity for standardized ML-DSA parameter sets.

Parameter set	ℓ	γ_1	$w = \log_2(2\gamma_1)$	Packed $N = 32\ell w$ (bytes)	Capacity $N - 32$ (bytes)
ML-DSA-44	4	2^{17}	18	2,304	2,272
ML-DSA-65	5	2^{19}	20	3,200	3,168
ML-DSA-87	7	2^{19}	20	4,480	4,448

4.3 Abort probability and preservation of the baseline abort loop

The construction introduces no payload-dependent acceptance condition, the abort predicate is exactly the FIPS 204 predicate applied to the derived transcript. In the random-function idealization for $\mathsf{XOF}_{K_{\mathsf{sc}}}$, the packed container B in Algorithm 1 is uniformly distributed for each attempt: r is uniform and S is pseudorandom, and XOR by a fixed string preserves uniformity. Since baseline FIPS 204 uses pseudorandom bytes (via its XOF) to feed the same $\mathsf{BitUnpack}(\cdot, \gamma_1 - 1, \gamma_1)$ mapping, the induced distribution on $\mathbf{y}$ is computationally indistinguishable from uniform given the PRF assumption on $\mathsf{XOF}_{K_{\mathsf{sc}}}$, and matches the baseline in the ideal model. Consequently, the abort probability and the distribution of the retry counter κ are unchanged in that model.

In the computational setting, any difference in abort behavior is bounded by the advantage of distinguishing $\mathsf{XOF}_{K_{\mathsf{sc}}}$ from random (and the corresponding pseudorandomness assumptions already used to justify the baseline $\mathsf{ExpandMask}$ sampler).

4.4 Public detectability

A public warden sees (pk, M, σ) and may adaptively query signatures. The distinguishing claim for the channel is: *under the PRF security of the instantiated keyed XOF $\mathsf{XOF}_{K_{\mathsf{sc}}}$ and the pseudorandomness idealizations for $\mathsf{ExpandMask}$ used in the FIPS 204 security analysis, signatures produced by $\mathsf{SC.Sign}$ are computationally indistinguishable from baseline FIPS 204 signatures to any PPT distinguisher without K_{sc} and $\mathbf{s}_1$.* With the reference SHAKE256 instantiation, this assumption is that the sponge with a secret-prefix input acts as a secure PRF keyed by K_{sc}, distinguishing the channel reduces to distinguishing SHAKE256 in that setting. This is the same modeling stance as in [11], specialized to the FIPS 204 byte-level sampler interface: see [1] for the baseline idealization.

Operational detection and auditing approaches. Beyond generic black-box statistical testing, meaningful detection typically requires either *side information* about the signer's randomness or a *trusted recomputation path*. In particular: (i) in the default hedged mode, a public warden is limited to monitoring coarse observables (e.g., signature-size/format violations, latency and retry-count distributions, and lightweight per-byte tests on σ) which are useful as regression checks against buggy subversions but do not rule out well-designed channels, (ii) a key-holding auditor can make detection much stronger by adopting the optional *deterministic* signing variant of FIPS 204 (set $\mathsf{rnd} = \{0\}^{32}$) and verifying that signatures are reproducible under a trusted implementation, or by requiring caller-supplied rnd values to be externally generated and logged/attested so that the

expected ExpandMask transcript can be recomputed. These operational controls complement in-band *keyed* detection (the magic‖tag approach above) and mitigation-by-sanitization (reverse-firewall wrappers).

4.5 Stripping/sanitization

An external party without signing secrets cannot re-randomize an ML-DSA signature while preserving validity, as the scheme is not publicly rerandomizable. A key-holding defender can sanitize by re-signing M using a trusted implementation or by placing a reverse-firewall-style wrapper at signing time [17, 18], designing robust sanitizers is outside the scope of this note.

5 Comparison point: SLH-DSA has a much smaller obvious embedding surface

For context, FIPS 205 SLH-DSA signatures explicitly include an n-byte randomization string R as their first field (with $n \in \{16, 24, 32\}$ depending on the parameter set) [2]. In a comparable signer controls randomness setting, a straightforward standards-aligned embedding point is therefore $n \in \{16, 24, 32\}$ bytes per signature, in contrast to the kilobyte-scale packed-commitment surface in FIPS 204 ML-DSA.

FN-DSA/Falcon (FIPS 206 draft). A second NIST-selected post-quantum signature design is Falcon (often referred to as FN-DSA in the context of the forthcoming FIPS 206 draft standard). Falcon follows a hash-then-sign paradigm and, in its current public specification, prepends a *uniform 320-bit salt* to the message before hashing ($r \in \{0,1\}^{320}$) [4]. In a signer controls randomness setting, this salt is an explicit, standards-aligned embedding surface of 40 bytes per signature. This is qualitatively closer to SLH-DSA's explicit R field than to ML-DSA's kilobyte-scale packed-commitment interface.

General remark on secret-key-assisted recovery. The recovery step $\mathbf{y} = \mathbf{z} - c\mathbf{s}_1$ is an instance of a general Schnorr/Fiat–Shamir pattern: whenever the published response is an additive blend of an ephemeral commitment and a secret-key term, any party holding the relevant secret can reconstruct the commitment. What is more specific to FIPS 204 is that (i) accepted-signature bounds make the reconstruction of $\mathbf{y}$ unambiguous as centered representatives, and (ii) the standardized $(\gamma_1 - 1, \gamma_1)$ packing has full support, giving a clean fixed-length *byte* interface for embedding. For other PQC candidates, similar channels may exist but the practical bandwidth depends on the size of the recoverable commitment and on whether the scheme's encodings/compressions leave unused code points (which can complicate byte-level embedding without introducing bias).

6 Implementation notes

Patch point. Hook precisely at the byte-to-BitUnpack$(\cdot, \gamma_1 - 1, \gamma_1)$ boundary: replace the input bytes with the masked container from Algorithm 1 and leave challenge derivation, response/hint computation, and the abort loop unchanged.

Reuse packing code. Operate on the packed byte arrays and call the existing BitUnpack/BitPack routines: reimplementing bit encodings risks endianness and constant-time bugs and can introduce bias.

Keying and API exposure. Instantiate $\mathsf{XOF}_{K_{\mathsf{sc}}}$ with a keyed construction. The reference build uses SHAKE256 as a keyed sponge, absorbing "MLDSA-SC" $\|$ K_{sc} $\|$ μ $\|$ r and squeezing the payload mask: standardized keyed variants (KMAC256 or keyed cSHAKE256 with the same domain string) are drop-in replacements. Because SC.Sign takes an explicit payload, do not expose it to untrusted callers in subversion-resistant deployments.

Side channels. Implement the keyed-XOF expansion, XOR masking, and packing/unpacking in constant time and with constant-memory-access patterns, matching the baseline ML-DSA posture.

Reference patch (C library). Implemented under `MLD_CONFIG_SC` (randomized API required). The SC sampler is inserted at the byte-to-BitUnpack boundary, a keyed-XOF embedder/extractor is added, and test-only entry points `crypto_sign_signature_sc` and `mld_sc_extract` are exposed. A small stats struct records total attempts and per-reason rejects (`z`, `w0`, `h`, `hint`) inside the abort loop.

Test harness and artifacts. The harness in `test/test_sc.c` reads workload parameters from the environment (`MLD_SC_NSIG`, `MLD_SC_NSTAT`, `MLD_SC_NDIST`). Helper scripts in `scripts/` post-process logs and plots, and outputs live under `test/results/sc/` (CSV summaries, plots, logs). Code and artifacts are available in the `mldsa-native-sublime` fork [3].

7 Experimental validation summary

We executed the minimal validation plan using the reference patch on ML-DSA-44/65/87. For round-trip correctness, we generated $N_{\text{sig}} = 500$ signatures per parameter set and payload lengths $L \in \{0, 16, 32, 64, 256, 1024, \mathsf{cap} - 32, \mathsf{cap}\}$, all signatures verified and all extracted payloads matched. Boundary checks reject $L > \mathsf{cap}$, and baseline signatures do not decode the chosen payload, tampering with a channel signature causes extraction to fail when verification is enabled.

For abort-rate statistics, we measured κ histograms and per-reason reject counters with $N_{\text{stat}} =$ 20,000 and 100,000 per parameter set. Two-sample KS tests on the bucketed κ CDFs show small D-statistics (max $D = 0.0112$ at 20k samples, max $D = 0.0027$ at 100k), and the measured mean-attempt differences are small. We record κ exactly for $1 \le \kappa \le 15$ and bucket all $\kappa \ge 16$ into a single tail bin. For distribution sanity, we performed chi-square tests on selected signature byte positions with $N_{\text{dist}} = 20{,}000$ and 100,000 per parameter set, no obvious anomalies were observed. We also report per-reason reject rates, tail probability $\Pr[\kappa \ge 16]$, and mean-attempt confidence intervals derived from the binned histograms. Selected plots appear in Table 4, and the full plot inventory is listed in Appendix A.

Reproducibility. Built with `OPT=1` and `EXTRA_CFLAGS=-DMLD_CONFIG_SC`.
Commit `f529ac35ab6744d110b6a16af2e406935fa7214a`.
Toolchain `gcc 15.2.0` on Linux `Ubuntu 25.04` (x86_64).
Paper workloads used `MLD_SC_NSIG=500` and set `MLD_SC_NSTAT=100000` and `MLD_SC_NDIST=100000`, running `./test/build/mldsa44/bin/test_sc44`.
The analogous `test_sc65`/`test_sc87` with `MLD_SC_NSTAT=20000` and `MLD_SC_NDIST=20000` yielded the 20k runs.

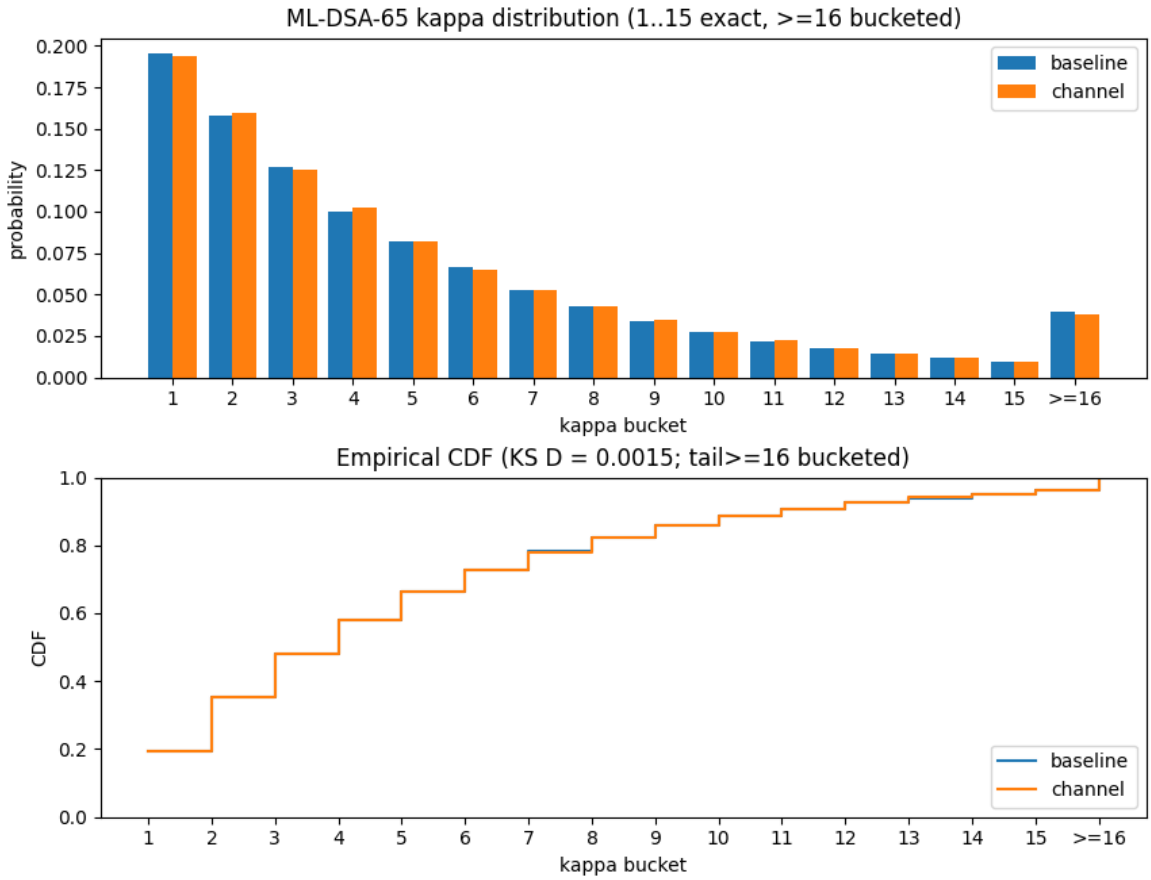

Figure 1: Kappa histogram and CDF for ML-DSA-65 (100k baseline vs channel).

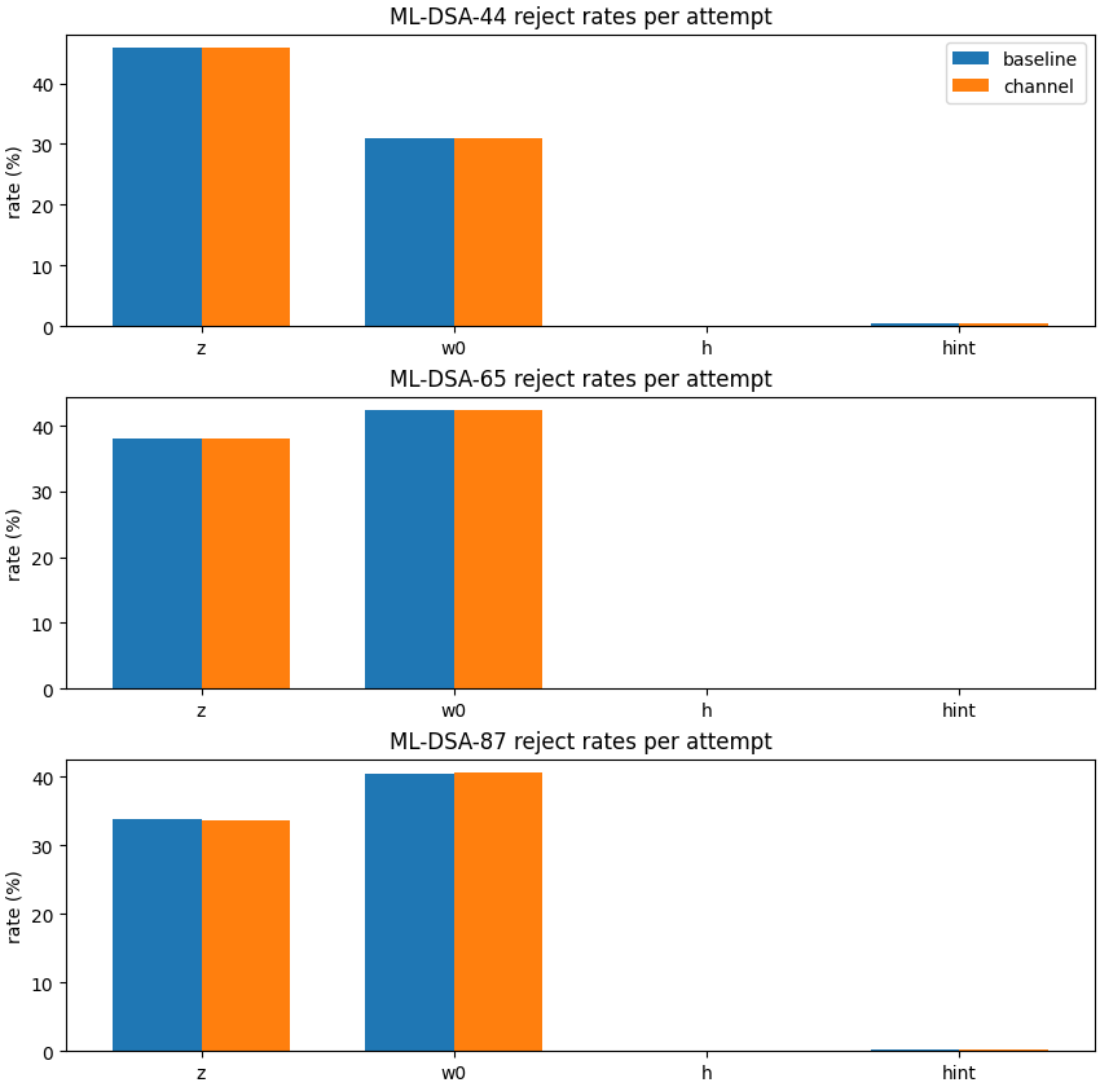

Figure 2: Per-attempt reject rates by reason (100k).

Table 3: Key abort statistics (100k samples per parameter set). Means are attempts per signature, tail is $\Pr[\kappa \geq 16]$, rates over attempts.

Param	Mode	Mean	Tail	Reject rate	N_{stat}
44	base	4.374	2.08%	77.14%	100,000
44	chan	4.376	2.05%	77.15%	100,000
65	base	5.133	3.93%	80.52%	100,000
65	chan	5.123	3.84%	80.48%	100,000
87	base	3.905	1.17%	72.70%	100,000
87	chan	3.887	1.16%	72.71%	100,000

7.1 Selected plots

Table 4: Selected validation plots (PDF format). Paths are relative to https://github.com/idrassi/mldsa-native-sublime/test/results/sc. Full inventory in Appendix A.

ID	Dataset	Param	Description	File (PDF)
M1	100k	44	Kappa histogram + CDF (baseline vs channel)	`plots_100k_pdf/kappa_cdf_mldsa44.pdf`
M2	100k	65	Kappa histogram + CDF (baseline vs channel)	`plots_100k_pdf/kappa_cdf_mldsa65.pdf`
M3	100k	87	Kappa histogram + CDF (baseline vs channel)	`plots_100k_pdf/kappa_cdf_mldsa87.pdf`
M4	20k vs 100k	65	Overlay CDFs (baseline 20k vs 100k; channel 20k vs 100k)	`plots_overlay_pdf/kappa_cdf_overlay_mldsa65.pdf`
M5	100k	all	Reject rates per attempt by reason	`plots_extra_100k_pdf/reject_rates_opt1_100k.pdf`
M6	100k	all	Tail probability $\Pr[\kappa \geq 16]$	`plots_extra_100k_pdf/kappa_tail_opt1_100k.pdf`
M7	100k	all	Mean attempts with 95% CI (binned tail)	`plots_extra_100k_pdf/mean_ci_opt1_100k.pdf`

Acknowledgments

This note is derived from the framework of Galteland and Gjøsteen [11], any errors are ours.

A Full plot inventory

Table 5: Summary of SC validation plots (PDF format). Paths are relative to https://github.com/idrassi/mldsa-native-sublime/test/results/sc. PNG and SVG variants are available in sibling `plots_*/` directories.

ID	Dataset	Param	Description	File (PDF)
P1	20k	44	Kappa histogram + CDF (baseline vs channel)	`plots_20k_pdf/kappa_cdf_mldsa44.pdf`
P2	20k	65	Kappa histogram + CDF (baseline vs channel)	`plots_20k_pdf/kappa_cdf_mldsa65.pdf`
P3	20k	87	Kappa histogram + CDF (baseline vs channel)	`plots_20k_pdf/kappa_cdf_mldsa87.pdf`
P4	100k	44	Kappa histogram + CDF (baseline vs channel)	`plots_100k_pdf/kappa_cdf_mldsa44.pdf`
P5	100k	65	Kappa histogram + CDF (baseline vs channel)	`plots_100k_pdf/kappa_cdf_mldsa65.pdf`
P6	100k	87	Kappa histogram + CDF (baseline vs channel)	`plots_100k_pdf/kappa_cdf_mldsa87.pdf`
P7	20k vs 100k	44	Overlay CDFs (baseline 20k vs 100k; channel 20k vs 100k)	`plots_overlay_pdf/kappa_cdf_overlay_mldsa44.pdf`
P8	20k vs 100k	65	Overlay CDFs (baseline 20k vs 100k; channel 20k vs 100k)	`plots_overlay_pdf/kappa_cdf_overlay_mldsa65.pdf`
P9	20k vs 100k	87	Overlay CDFs (baseline 20k vs 100k; channel 20k vs 100k)	`plots_overlay_pdf/kappa_cdf_overlay_mldsa87.pdf`
P10	20k	all	Reject rates per attempt by reason	`plots_extra_20k_pdf/reject_rates_opt1_20k.pdf`
P11	20k	all	Tail probability $\Pr[\kappa \geq 16]$	`plots_extra_20k_pdf/kappa_tail_opt1_20k.pdf`
P12	20k	all	Mean attempts with 95% CI (binned tail)	`plots_extra_20k_pdf/mean_ci_opt1_20k.pdf`
P13	100k	all	Reject rates per attempt by reason	`plots_extra_100k_pdf/reject_rates_opt1_100k.pdf`
P14	100k	all	Tail probability $\Pr[\kappa \geq 16]$	`plots_extra_100k_pdf/kappa_tail_opt1_100k.pdf`
P15	100k	all	Mean attempts with 95% CI (binned tail)	`plots_extra_100k_pdf/mean_ci_opt1_100k.pdf`

References

[1] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard (FIPS 204)*. Federal Information Processing Standards Publication 204, Aug 2024. DOI: 10.6028/NIST.FIPS.204.

[2] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard (FIPS 205)*. Federal Information Processing Standards Publication 205, Aug 2024. DOI: 10.6028/NIST.FIPS.205.

[3] Mounir Idrassi. *mldsa-native-sublime: Subliminal-channel reference implementation for ML-DSA*. Available at https://github.com/idrassi/mldsa-native-sublime.

[4] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU* (Specification v1.2). Oct 2020. Available at https://falcon-sign.info/falcon.pdf.

[5] National Institute of Standards and Technology. *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash* (SP 800-185). Available at https://doi.org/10.6028/NIST.SP.800-185.

[6] Gustavus J. Simmons. The Prisoners' Problem and the Subliminal Channel. In David Chaum (ed.), *Advances in Cryptology – CRYPTO '83*, pages 51–67. Plenum Press, 1984.

[7] Gustavus J. Simmons. Subliminal Channels: Past and Present. *European Transactions on Telecommunications*, 5(4):459–474, 1994.

[8] Yvo Desmedt. Subliminal-Free Authentication and Signature (Extended Abstract). In C. G. Günther (ed.), *Advances in Cryptology – EUROCRYPT 1988*, LNCS 330, pages 23–33. Springer, 1988.

[9] Alexander Hartl, Robert Annessi, and Tanja Zseby. A Subliminal Channel in EdDSA. Preprint, 2017. Available at https://www.annessi.net/data/2017-subliminal_preprint.pdf.

[10] Alexander Hartl, Robert Annessi, and Tanja Zseby. Subliminal Channels in High-Speed Signatures. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* (JoWUA), 9(1):26–49, 2018.

[11] Herman Galteland and Kristian Gjøsteen. Subliminal Channels in Post-Quantum Digital Signature Schemes. Cryptology ePrint Archive, Report 2019/574, 2019. Available at https://eprint.iacr.org/2019/574.

[12] NIST Computer Security Resource Center. Official comments on CRYSTALS-Dilithium (round-1). Notes that $\mathbf{s}_1$ enables alternative signing/forgery procedures. Available at NIST CSRC round-1 official comments.

[13] Adam Young and Moti Yung. Kleptography: Using Cryptography Against Cryptography. In *Advances in Cryptology – EUROCRYPT '97*, LNCS 1233, pages 62–74. Springer, 1997.

[14] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-Resilient Signature Schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS 2015)*, pages 364–375. ACM, 2015. DOI: 10.1145/2810103.2813635.

[15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signatures: Definitions, constructions and applications. *Theoretical Computer Science*, 820:91–122, 2020. DOI: 10.1016/j.tcs.2020.03.021.

[16] Sebastian Berndt and Maciej Liśkiewicz. Algorithm Substitution Attacks from a Steganographic Perspective. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017)*, pages 1649–1665. ACM, 2017. Preprint: arXiv:1708.06199.

[17] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic Reverse Firewalls. In *Advances in Cryptology – EUROCRYPT 2015*, LNCS 9056, pages 657–686. Springer, 2015. DOI: 10.1007/978-3-662-46803-6_22. Full version: Cryptology ePrint Archive, Report 2014/758.

[18] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message Transmission with Reverse Firewalls—Secure Communication on Corrupted Machines. In *Advances in Cryptology – CRYPTO 2016*, LNCS 9814, pages 341–372. Springer, 2016. DOI: 10.1007/978-3-662-53018-4_13.