



// PUBLIC • SOURCE-LEVEL AUDIT

Security Audit.

CROSS post-quantum signature verification

TARGET	CROSS-signature/CROSS-implementation
BUILD	main, commit b2a84251 (pre-fix)
DATE	September 2026
PREPARED BY	Eva Crystal (Oxiviél), XSource Security
CLASSIFICATION	Public

Executive summary

XSource Security reviewed the verification path of CROSS, a code-based post-quantum signature scheme in the NIST additional-signatures process. The review was source-level and self-funded, and the result was disclosed to the maintainers under coordinated disclosure.

One finding was confirmed and is now public. `crypto_sign_open()`, the combined open-and-verify entry point the reference implementation declares in its public header, subtracts the signature size from the caller-supplied signed-message length without first checking that the length is at least that large. The subtraction is unsigned. A signed message shorter than one signature makes the result wrap to a value near 2^{64} , and the function hands that value straight to `memcpy`. Verification input is untrusted by definition, so no privilege, no key material and no prior state are needed to reach it: delivering a short buffer is the whole precondition.

The defect was found in the copy of CROSS vendored inside liboqs and traced to the upstream reference implementation, where it sits at the same two lines. Upstream fixed it in commit `fc6b7e7` with exactly the guard that was recommended, published it as GHSA-w72c-hgx8-p7cv, and the finding is now CVE-2026-41509 at 9.8 Critical (CVSS 3.1, NVD).

Findings confirmed	1
Severity	9.8 Critical (CVSS 3.1, NVD)
Affected	every commit before <code>fc6b7e7</code>
Status	Published, fixed upstream and downstream

Methodology

The review followed the standard XSource audit method, with two points specific to a vendored cryptographic primitive:

- **Sweep the library, then rank on evidence.** CROSS was one of the algorithm families read in a single pass over liboqs `0.15.0-19-gd1d8447`. Ten findings came out of that pass. This one was ranked first for submission because it was the only one with a reproduced crash, not because it looked worst on paper.
- **The vendored copy is not the identity.** The defect was found in liboqs, so the first question after confirming it was whose code it is. It is upstream's, present in `CROSS-signature/CROSS-implementation` at the same two lines, and that is where it was reported and where it is fixed. The liboqs path is recorded here as where it was found, not as the affected product.
- **Reachability separated from presence.** Code being present in a build is not the same as code being reachable through a shipped API. liboqs carries the defective function in all 36 CROSS

variant directories and never calls it, which is stated below rather than left for a reader to discover.

- **Reproduced, not inferred.** The finding is backed by a standalone PoC, built against the audited tree, that aborts under AddressSanitizer. It was re-run for this report on 2026-09-22 and still reproduces.
- **Limits stated upfront.** What was not measured is named in the finding rather than left implicit, so severity is judged on what was actually shown.

Findings

ID	FINDING	SEVERITY	STATUS
F01	Integer underflow of <code>mLen</code> in <code>crypto_sign_open()</code> leads to a buffer overflow	9.8 Critical (CVSS 3.1, NVD)	Fixed in <code>fc6b7e7</code> , published as CVE-2026-41509

F01 • Integer underflow in `crypto_sign_open()` leads to a buffer overflow

Severity. 9.8 Critical (CVSS 3.1, NVD). Vector `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`, scored by NIST NVD as the primary metric on a record in state Analyzed, published 2026-05-08. The finding was filed as CWE-191, integer underflow, leading to CWE-122; the advisory classifies it as CWE-121 and CWE-122. The GitHub advisory labels it Critical and publishes no vector of its own.

Affected. `CROSS-signature/CROSS-implementation`, file `ReferenceImplementation/lib/sign.c`, every commit before `fc6b7e78cdf789bb5c395a81dc601356f1383da0`. The immediate parent of the fix is `b2a84251c83840e016aaaabc5ca82fafc91ee5ef`, dated 2025-07-31, which is the tree this report is written against. Downstream, liboqs vendors the same function into all 36 CROSS variant directories at `0.15.0-19-gd1d8447`; all 36 carry the defect, and none of them is reached by liboqs itself.

Preconditions. The caller uses the combined NIST API `crypto_sign_open()` and passes a signed message shorter than `sizeof(CROSS_sig_t)`. Nothing else: no valid key, no valid signature, no session, no prior call. `crypto_sign_open()` is declared in `ReferenceImplementation/include/api.h:64` and is the function the shipped KAT harness drives at `AdditionalImplementations/KAT_Generation/lib/PQCgenKAT_sign.c:159`, so it is the supported way to verify with this implementation, not an internal helper.

Root cause

```
// Reference_Implementation/lib/sign.c, before fc6b7e7
int crypto_sign_open(unsigned char *m,
                    unsigned long long *mlen,
                    const unsigned char *sm,
                    unsigned long long smlen,
                    const unsigned char *pk)
{
    /* verify returns 1 if signature is ok, 0 otherwise */
    *mlen = smlen-(unsigned long long) sizeof(CROSS_sig_t);    // :84

    memcpy((unsigned char *) m,
           (const unsigned char *) sm, (size_t) *mlen);        // :86
```

`smlen` and `*mlen` are both unsigned. When `smlen < sizeof(CROSS_sig_t)`, line 84 wraps instead of going negative, and line 86 copies that wrapped length. There is no check anywhere between the function's first statement and the copy. The signature check itself, `CROSS_verify()`, runs after the copy and cannot prevent it.

The liboqs copy is textually different, using `size_t` rather than `unsigned long long` and differing in whitespace, and is identical in behaviour: the same two statements sit at `src/sig/cross/upcross_cross-rsdp-128-fast_clean/sign.c:84` and `:86` in the audited tree.

Reachability

Reachable by anyone who supplies the buffer being verified. There is no authentication step before it, because it is itself part of the step that establishes authenticity.

It is worth being precise about who is exposed, because the two obvious consumers differ:

CONSUMER	CALLS	EXPOSED
Reference implementation, public API	<code>crypto_sign_open()</code> , <code>api.h:64</code>	Yes
Shipped KAT harness	<code>crypto_sign_open()</code> , <code>PQCgenKAT_sign.c:159</code>	Yes
liboqs <code>OQS_SIG_verify</code>	<code>crypto_sign_verify()</code> , which takes <code>siglen</code> separately and checks it	No

liboqs routes every CROSS verification through the detached-signature entry point instead, at `src/sig/cross/sig_cross_rsdp_128_fast.c:82`, `:85` and `:89`, and the same pattern holds for the other parameter sets. So a liboqs user is not exposed through liboqs. Anything consuming the reference implementation directly, which is how the repository ships and how its own test harness drives it, is.

Evidence

A standalone PoC, `poc_cross_underflow.c`, links against the audited build and calls the vulnerable function with `smLen = 0`, the smallest value that triggers the wrap. It needs no key, no signature and no network:

```
gcc -I <build>/include -L <build>/lib -fsanitize=address -g -O1 \
    poc_cross_underflow.c -loqs -lcrypto -lm -o poc_cross_underflow
```

The build is the `build-clang-01` configuration of the audited tree, retained alongside the PoC so the result is re-runnable rather than asserted. Re-run on 2026-09-22, frames elided:

```
==199207==ERROR: AddressSanitizer: negative-size-param: (size=-18432)
    #0 memcpy
    #1 PQCLEAN_CROSSRSDP128FAST_CLEAN_crypto_sign_open sign.c:86
    #2 main poc_cross_underflow.c:46

0x7b6f619e0010 is located 0 bytes inside of 16-byte region
allocated by thread T0 here:
    #1 main poc_cross_underflow.c:32
```

This is the same crash, at the same line, with the same size, as the run recorded when the finding was written on 2026-03-03.

Impact

`sizeof(CROSS_sig_t)` is 18432 bytes for this parameter set, so `smLen = 0` produces `*mLen = 0 - 18432 = 0xFFFFFFFFFFFFB800`. That is the length passed to `memcpy`, which reads from a 16-byte source region and writes into a 64-byte destination region. Both bounds are crossed on the first call.

What is shown is an out-of-bounds read and an out-of-bounds write of attacker-influenced length, from and into heap memory, during signature verification, reached with a single short buffer and nothing else. Under a sanitizer the process aborts. Without one, `memcpy` with a length near `SIZE_MAX` walks until it faults, so the practical outcome on an unsanitized build is a crash of the verifier after an unbounded amount of adjacent memory has been overwritten.

What is not shown is control of that write. No work was done on whether the copy can be steered into a useful target before it faults, and this report does not claim code execution.

Stated limits

Three limits were held to while writing this up, so severity rests on what was actually demonstrated:

- One variant was executed. The PoC ran against `rsdp-128-fast_clean`. The other 35 vendored variants, and the upstream reference tree, were confirmed by reading the source, not by running them.

- Upstream was confirmed by source, not by execution. The pre-fix upstream file was read at `b2a84251` and carries the defect at the same two lines; no upstream build was made.
- Exploitability beyond the out-of-bounds copy was not attempted. See Impact.

Remediation

One check, before the subtraction:

```
// Reference_Implementation/lib/sign.c:83-86, added by fc6b7e7
if (smlen < (size_t) sizeof(CROSS_sig_t)) {
    return -1;
}
```

This is what was recommended and it is, line for line, what shipped. The commit message is "add a check on the minimum smlen", four lines added, one file touched, and it is on `main`.

Two things are worth saying about the shape of the fix rather than its content. It closes the instance at the only place the length is known before it is used, which is the right place. It does not close the class: the same combined open-and-verify API shape, an unsigned length arriving from the caller and being reduced by a constant before use, recurs across NIST-API reference implementations, and each one needs the same check written into it separately.

Resolution

DATE	EVENT
2026-03-03	Found and reproduced under AddressSanitizer during a full-library pass over liboqs <code>0.15.0-19-gd1d8447</code>
2026-03-07	Reported to liboqs as issue #2409; the date is the one GHSA-w72c-hgx8-p7cv states
2026-03-23	Fixed upstream in <code>fc6b7e7</code> , "add a check on the minimum smlen"
2026-03-24	Published as GHSA-w72c-hgx8-p7cv, Critical, no CVSS vector given
2026-05-08	CVE-2026-41509 published; NVD scored it 9.8 Critical (CVSS 3.1)
2026-09-12	Fix confirmed downstream: liboqs <code>0.16.0-44-g1409b0dad</code> carries the guard in all 36 vendored copies

The advisory credits the finder as `iiviel`. The CVE record carries no credits entry.

Coverage

Coverage below is the CROSS part of the pass that produced F01. The CROSS areas that produced nothing are listed alongside the ones that did, because both are part of what an audit delivers.

AREA	EXAMINED	RESULT
CROSS, all 36 vendored variants	Yes	F01, the subject of this report
Upstream CROSS reference tree	Yes	Same defect at the same two lines as the vendored copy; fix present on <code>main</code>
Exploitability of F01 beyond the out-of-bounds copy	No	Not attempted. No work on controllability, target selection or code execution
The other 35 CROSS variants at runtime	No	Read, not executed
Functional correctness and KAT conformance	No	Out of scope. No functional or conformance testing was done
Downstream consumers other than liboqs	No	Not surveyed. Whether any packaged distribution exposes <code>crypto_sign_open()</code> to untrusted input was not established

Conclusion

F01 is a missing length check on the one function in the scheme whose entire input is untrusted. It took one comparison to fix, and upstream wrote exactly that comparison. The reason it survived to a CVE is not difficulty, it is that the combined open-and-verify API is the part of a signature implementation that gets read least, because it looks like plumbing rather than cryptography.

Appendix A • Reporting standard

Five conditions every XSource finding meets before it is written up:

- **Reproduced, not inferred.** Triggered against the agreed target in the agreed environment, or, where static analysis is the agreed method, reproduced as a standalone proof with the limits stated.
- **Reachable from a real position.** The attacker position is named and was actually held, or, where static analysis is the agreed method, the path to it is traced from source end to end with the limits stated.

- **Impact shown, not asserted.** Which data, whose data, or which action, with the evidence attached.
- **Checked against design and backlog first.** Anything that might be intended behaviour, or already tracked, is raised before it is written up as a defect.
- **Fixable as written.** Remediation names the location and the change, and says so plainly where the fix is a design decision rather than a patch.

XSource Security · Eva Crystal (Oxiviel) · Christchurch, New Zealand · <https://xsourcesec.com>
Findings are credited as Oxiviel, iiviel or Eva Crystal (XSource Security).