

An Analysis of the ITU-T Multiple Cryptographic Algorithms X.509 Extensions for PQC Migration

Falko Strenzke 

MTG AG, Darmstadt, Germany

Abstract. The 2019 edition of ITU-T X.509 adds three certificate extensions, colloquially called the Catalyst extensions, that let a single certificate carry a second, alternative public key and issuer signature beside the native ones, as a backwards-compatible path for migrating a public-key infrastructure to post-quantum cryptography (PQC). This paper collects the known problems of this multiple cryptographic algorithms (MCA) mechanism and adds several findings of its own. The central and previously known weakness is that a quantum attacker who recovers a single traditional certification-authority key can replace any certificate in a chain by a traditional-only one that a relying party still accepts. We give a precise attack model with a sub-CA and an end-entity-forgery variant to emphasize the inherent problem and propose three mitigating policies for relying parties which offer different degrees of protection and interoperability. As an example, we verify empirically that the susceptibility to this downgrade attack is present in the wolfSSL cryptographic library as a natural consequence. Our further findings concern the construction itself and details of the specification. For one, we analyze the formal violations of EUF-CMA that arise due to the possibility of re-purposing the extension signature to the established signed data format in X.509, which is formally manifest as soon as the alternative key in the extension is also certified as a native key. We show that several further aspects in the ITU-T mechanism remain undefined, which potentially leads to divergent behaviour of clients, and point out that a certificate accepted on its alternative signature alone, as allowed per ITU-T specification, can be invalid under RFC 5280 and potentially exposes an application to arbitrary injected data and even modified signatures chaining it to a certificate chosen by a non-quantum attacker in the positively validated certificate chain.

Keywords: PQC · X.509 · multi-algorithm · digital signature

1 Introduction

The migration of X.509 public-key infrastructures to post-quantum cryptography (PQC) faces a chicken-and-egg problem: certification authorities (CAs) cannot switch their signature algorithms before all relying parties support them, while relying parties' implementations that support PQC certificates cannot adopt a strict "require-PQC" policy until stable PQC CAs are available. The 2019 edition of ITU-T X.509 [Int19] tries to address this with the introduction of the Multiple Cryptographic Algorithms (MCA) mechanism, which rests on three newly introduced certificate extensions that let a single certificate carry a second public key and a second issuer signature. The apparent intent is that one certificate serves migrated and non-migrated relying parties alike. This paper collects the known problems of the MCA mechanism and adds a number of findings of our own. We first review the stages of a PQC migration that a backwards-compatible extension of this kind could support (Section 1.1), and then describe the extensions as specified,

E-mail: falko.strenzke@mtg.de (Falko Strenzke)

including the changes made by the corrigendum of 2026 (Sections 1.2 to 1.5). Section 2 shows that the extensions give a relying party no way to insist on PQC verification, so that a quantum attacker downgrades any MCA hierarchy to a traditional one, leading to a direct authentication bypass. We provide a set of three mitigating policies with different trade-offs between interoperability and security against a quantum-attacker. Section 3 shows that the alternative signature can be re-encoded as a stand-alone signed object under the CA's alternative key, a formal violation of existential unforgeability that, however, in all likelihood does not lead to an exploitable vulnerability. Section 4 shows that a certificate accepted on its alternative signature alone can be invalid under RFC 5280 and that the native signature can be modified arbitrarily, with potential negative effects on security. Section 5 collects the points of verifier behaviour the specification leaves unspecified, and Section 6 analyses the two existing implementations in FOSS and confirms the downgrade against them. Section 7 gives our conclusions.

1.1 Background on the PQC Migration Scenario

Before turning to the ITU-T extensions themselves, we set out what a backwards-compatible PQC extension could achieve, so that we can later judge the MCA mechanism against it. During the migration of a PKI to post-quantum cryptography one can distinguish three stages, each offering a different benefit to a relying party.

1. *Opportunistic PQC validation.* A relying party that supports the new algorithms verifies a post-quantum signature when one is present, but this has no security effect: a quantum attacker can still downgrade the relying party exactly as if only a traditional certificate chain were used. The only value at this stage is given if it is possible to collect metrics about the PQC readiness of the ecosystem.
2. *PQC opt-in by the authenticating party.* Once the relying party learns that the authenticating party has opted in – the root or a sub-CA, meaning it will always provide a fully post-quantum path, or the individual end entity – it stops accepting purely traditional signatures and always requires a post-quantum signature of the respective end-entity. This is the first stage with a security benefit against a quantum attacker.
3. *Mandatory PQC.* Post-quantum signatures are required unconditionally.

The Chrome team's roadmap for post-quantum authentication in the Web PKI [Ben26, Chr26] follows exactly this model, with one additional final phase [Ben26]. Its first two stages correspond to our first two stages, its fourth stage to our third. Its third stage is a stricter version of the opt-in stage, where PQC CAs are mandatory but traditional end-entity keys are still allowed. In this phase, a relying party that is PQ-ready is safe against a downgrade due to the rule that any PQC-CA issues only certificates for the qualitatively strongest key – in other words: if an end entity holds a PQC key, the PQC-CA only signs this key, not a potentially existing traditional one.

An important fact about the MCA mechanism is that it reaches none of these stages, as we detail in Section 2 with reference to prior literature. A quantum attacker downgrades any hierarchy to a purely traditional one (Section 2), so the mechanism offers no more than the optional verification of post-quantum signatures of stage 1. It does not even deliver any metrics or facilitate such a mechanism that would give that stage its value. Because the verification of the alternative signature is a feature buried inside the opaque path-validation routine, the existing interfaces do not expose which signatures were actually verified: a TLS server does not learn what the client verified, and even the client's own application has no means, through the typical path-validation interface, to determine the validated signatures. The mechanism therefore does not even yield an indication of the ecosystem's PQC readiness.

1.2 The ITU-T Multiple Cryptographic Algorithms Extensions

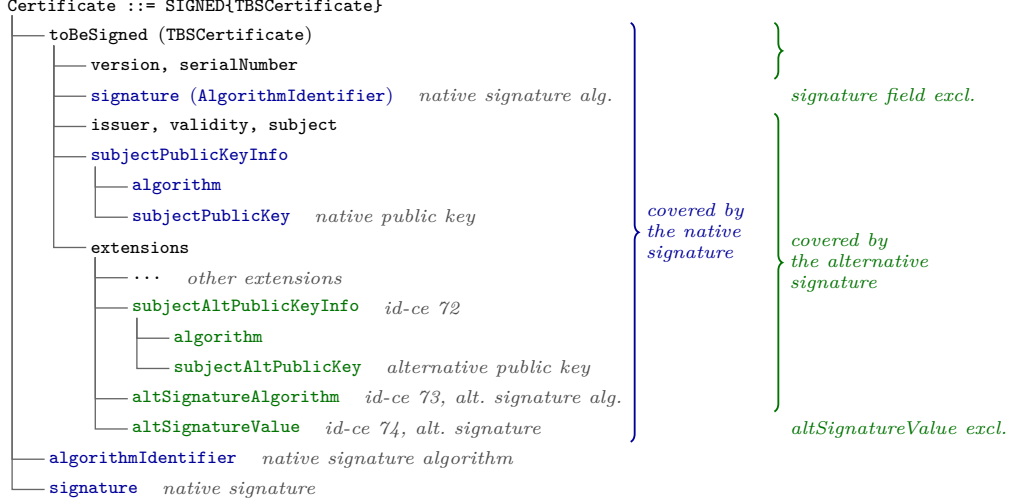
Clause 9.8 of ITU-T X.509 (10/2019) [Int19] defines three extensions “to provide capabilities for easy migration within a PKI from one set of cryptographic algorithms to another more suitable set of cryptographic algorithms”: the subject alternative public key information extension, the alternative digital signature algorithm extension, and the alternative signature value extension. Throughout this paper we refer to them as the *multiple cryptographic algorithms (MCA) extensions*, or the *MCA mechanism*. They are colloquially also known as the *Catalyst extensions*, after the name under which ISARA and others first proposed them to the IETF LAMPS working group in an Internet-Draft [TVF⁺18]. The 2019 edition of ITU-T X.509 took the extensions over from the draft, with the changes discussed in Sections 1.3 and 1.6. The draft’s final version of 2023 formally set the work aside, giving as reasons the large PQC key sent even to legacy clients and the lack of a generic encoding for multiple signatures [TVF⁺23]. The ITU-T specification is thus the only standard that defines them. A certificate carrying them is called a *multiple cryptographic algorithms public-key certificate* (clause 3.5.42). The extensions may be included in CA certificates and in end-entity certificates. The two signature-related extensions may also be used in CRLs and in authority validation lists (clause 9.8.1), and clause 17.7 extends their use to attribute certificates, AA certificates and attribute certificate revocation lists.

Listing 1 reproduces the ASN.1 of the three extensions together with the definition of the certificate itself, and Figures 1a and 1b show where the native and the alternative fields sit in the structure of one certificate and what each signature covers. The relevant native fields are `subjectPublicKeyInfo`, which holds the native public key, the `signature` field of `TBSCertificate` and the outer `algorithmIdentifier`, which both name the native signature algorithm, and the outer `signature`, which holds the native signature. The three extensions mirror exactly these fields: `SubjectAltPublicKeyInfo` has the same syntax as `SubjectPublicKeyInfo`, `AltSignatureAlgorithm` is an `AlgorithmIdentifier`, and `AltSignatureValue` is a bit string. In the intended PQC migration scenario the native fields would hold the traditional cryptographic artifacts, and the fields in the extensions those of a PQC algorithm.

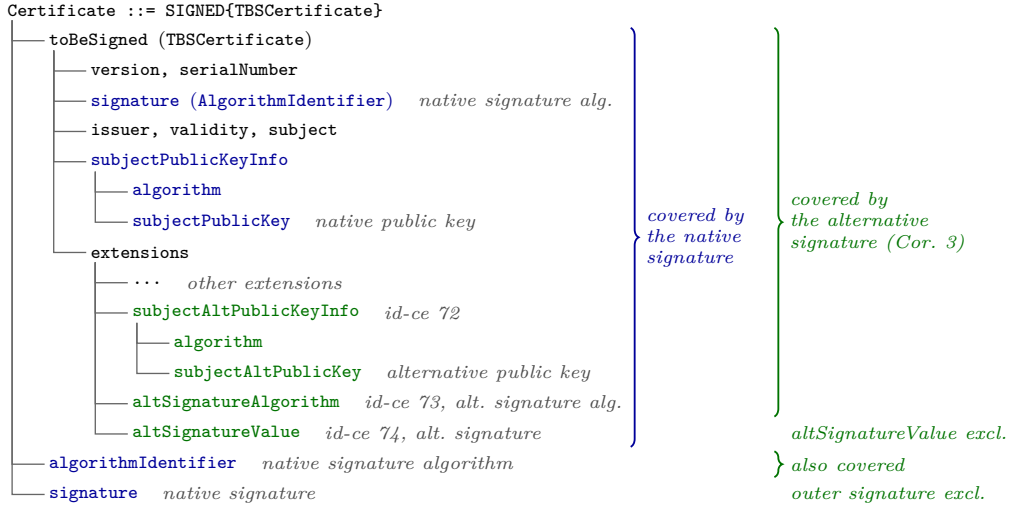
Note that Corrigendum 3 of 2026 (Section 1.4) removes two fields from the certificate ASN.1 profile that were reserved in the ITU-T specification but never put to use. Specifically, from the 2016 edition until Corrigendum 3 (2026) the `SIGNED` type itself carried two optional components for a second signature, `altAlgorithmIdentifier` and `altSignature`, which clause 6.2 nevertheless required to be absent. These two fields are removed by Corrigendum 3. The second signature is carried in the MCA extensions instead. In contrast to the removed mechanism, this has the advantage that a legacy implementation of a relying party that does not know them sees an ordinary certificate.

1.3 Generation and Verification of the Alternative Signature

Clause 7.2.2 specifies how the two signatures are computed. After listing what a certificate may hold, it states: “A public-key certificate that includes the alternative cryptographic algorithm extensions shall contain alternative public key information in the `subjectAltPublicKeyInfo` extension (see clause 9.8.2), an alternative signature algorithm value in the `altSignatureAlgorithm` extension (see clause 9.8.3) and an alternative signature value in the `altSignatureValue` extension (see clause 9.8.4).” Read strictly, this forbids a certificate that carries an alternative key but no alternative signature, and vice versa. We consider this wording an imprecision rather than a rule: clause 9.8.3 ties only `altSignatureAlgorithm` to `altSignatureValue`, the CRL variant of the mechanism (clause 7.10.3) uses the two signature extensions without any key extension, a CA that has no alternative key must still be able to certify a subject’s alternative key, and a CA that has one must be able to sign alternatively a certificate whose subject has only a native



(a) Structure of a multiple cryptographic algorithms public-key certificate in the reading of the 2019 version of the ITU-T specification [Int19]. Blue: native key and signature fields; green: the MCA extensions. The braces mark what each signature covers: the native signature covers the complete `toBeSigned` part including all three extensions; the alternative signature is shown in the 2019 reading of the signature coverage (Section 1.3), which is the `PreTBSCertificate` the Catalyst draft had defined and which Bouncy Castle and wolfSSL implement: it covers `toBeSigned` without its `signature` field and without the `altSignatureValue` extension. Figure 1b shows the Corrigendum 3 reading.



(b) The Corrigendum 3 reading of the signature coverage, as clarified by Corrigendum 3 (2026) of ITU-T X.509 (2019) [Int26]: the alternative signature covers the whole certificate except the outer `signature` value and the `altSignatureValue` extension, hence also both native algorithm identifiers. Compare Figure 1a.

Figure 1: Structure of a multiple cryptographic algorithms public-key certificate and the coverage of the two signatures under the 2019 reading and the Corrigendum 3 reading of the signature coverage.

```

Certificate ::= SIGNED{TBSCertificate}

SIGNED{ToBeSigned} ::= SEQUENCE {
    toBeSigned      ToBeSigned,
    algorithmIdentifier AlgorithmIdentifier{{SupportedAlgorithms}},
    signature        BIT STRING,
    ... }           -- alt components removed by Cor. 3 (2026)

TBSCertificate ::= SEQUENCE {
    version          [0] Version DEFAULT v1,
    serialNumber      CertificateSerialNumber,
    signature         AlgorithmIdentifier{{SupportedAlgorithms}},
    issuer            Name,
    validity          Validity,
    subject           Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    ...,
    extensions        [3] Extensions OPTIONAL }

subjectAltPublicKeyInfo EXTENSION ::= {
    SYNTAX            SubjectAltPublicKeyInfo
    IDENTIFIED BY      id-ce-subjectAltPublicKeyInfo }

SubjectAltPublicKeyInfo ::= SEQUENCE {
    algorithm          AlgorithmIdentifier{{SupportedAltPublicKeyAlgorithms}},
    subjectAltPublicKey BIT STRING }

altSignatureAlgorithm EXTENSION ::= {
    SYNTAX            AltSignatureAlgorithm
    IDENTIFIED BY      id-ce-altSignatureAlgorithm }

AltSignatureAlgorithm ::= AlgorithmIdentifier{{SupportedAltSignatureAlgorithms}}

altSignatureValue EXTENSION ::= {
    SYNTAX            AltSignatureValue
    IDENTIFIED BY      id-ce-altSignatureValue }

AltSignatureValue ::= BIT STRING

id-ce-subjectAltPublicKeyInfo OBJECT IDENTIFIER ::= {id-ce 72}
id-ce-altSignatureAlgorithm OBJECT IDENTIFIER ::= {id-ce 73}
id-ce-altSignatureValue OBJECT IDENTIFIER ::= {id-ce 74}

```

Listing 1: ASN.1 of the MCA extensions and of the certificate (ITU-T X.509 (10/2019) as corrected by Corrigendum 3, clauses 6.2, 7.2.1, 9.8 and Annex A; abbreviated).

key. The logical reading, which we adopt in the following and in Table 1, is therefore that `altSignatureAlgorithm` and `altSignatureValue` always go together, whereas the presence of `subjectAltPublicKeyInfo` is independent of the other two. Section 6 reports that wolfSSL has adopted that logical reading rather than the strict one (S5 to S9 in Table 2).

Further, “a CA generating a public-key certificate with the alternative cryptographic algorithms and alternative digital signature shall” (clause 7.2.2)

- compute the alternative signature over the DER encoding of the certificate from which what the clause calls “the signature component” and the `altSignatureValue` extension have been excluded, using the algorithm named in `altSignatureAlgorithm` (clause 7.2.2, quoted in full in Section 1.4) and, by clause 9.8.4, its alternative private key; and
- compute the native signature over the DER encoding of the certificate including all three extensions (clause 7.2.2).

The 2019 text leaves open what “the signature component” denotes: the `signature` field of `TBSCertificate`, which holds the native algorithm identifier, or the `signature` component of the outer `SIGNED` structure, which holds the native signature value. Corrigendum 3 of 2026 [Int26] settles it in favour of the latter (Section 1.4). The Catalyst draft, from

which the ITU-T text was derived, had defined the input as its `PreTBSCertificate`: the `TBSCertificate` with the `signature` field and the `altSignatureValue` extension removed, so that neither algorithm identifier nor either signature value is covered (Figure 1a). The draft’s final version gives a workflow reason for omitting the `signature` field: it “contains the `AlgorithmIdentifier` of the algorithm which will be used to sign the final certificate, and this value might not be known at the time that the alternative signature is calculated” [TVF⁺23]. The 2019 text admits a reading under which the coverage is exactly this, which we call the *2019 reading of the signature coverage* in the following. Under the second reading, which Corrigendum 3 fixes and which we call the *Corrigendum 3 reading*, the alternative signature covers the entire certificate except the outer signature value and the `altSignatureValue` extension: both native algorithm identifiers, the alternative public key and the alternative algorithm identifier are covered (Figure 1b). In both readings the alternative signature is computed first and the native signature last. Seen historically, it is thus ITU-T that departed from the draft’s definition. As Section 6 shows, both existing implementations follow the draft, i.e. the 2019 reading.

For verification, clause 7.2.2 distinguishes two kinds of relying parties. “A relying party that has not migrated to support alternative cryptographic algorithms and alternative digital signatures shall verify the native digital signature”, over the certificate as received. “A relying party that has migrated [...] shall verify the alternative digital signature”. To this end it shall decode the certificate, remove the signature component (the outer signature value, Section 1.4) and the `altSignatureValue` extension and re-encode the remainder in DER. The path processing procedure gives the corresponding processing rules in clause 12.5.1 a): if the relying party has not migrated, check that the signature verifies; if it supports alternative digital signatures, check that the alternative signature verifies. Clause 7.7 states the rule once more for the certification path as a whole, but with a condition that clause 12.5.1 a) lacks: only “if the public-key certificates in the certification path include multiple cryptographic extensions [*sic*]¹, and the relying party supports this extension”, shall the relying party “validate the alternative signature stored in the `altSignatureValue` extensions of the public-key certificates in the path”. It is this condition that leaves the alternative signature optional for the relying party (Section 2.1). CRLs follow the same scheme with the two signature extensions (clause 7.10.3). The ITU-T specification expects that “after the migration period” certificates are issued without the extensions and with the new algorithms in the base certificate (clause 7.2.2).

1.4 Corrigenda and Amendment to the 2019 Edition

The 2019 edition has since received three corrigenda and one amendment. Corrigendum 1 (10/2021), Corrigendum 2 (10/2023) [Int23] and Amendment 1 (10/2024, “Miscellaneous enhancements”) [Int24] do not touch the MCA mechanism; the amendment adds one directory attribute and updates the import lists of the ASN.1 modules. Corrigendum 3 (approved 2026-07-14, at the time of writing available only as an unedited prepublication) [Int26] changes it in three places; Appendix A shows the affected text as a marked-up diff.

- *Defect report 446* only addresses remaining artifacts from a previously conceived mechanism that was replaced by the alternative signature extensions. Specifically it removes the unused `altAlgorithmIdentifier` and `altSignature` components from the `SIGNED` type and deletes the identical sentence in clauses 7.2.1, 7.10.2, 11.3 and 14.2 that required them to be absent.
- *Defect report 450* clarifies the term “signature component” in clause 7.2.2 (discussed below).

¹The word “algorithm” is missing in the original.

- *Defect report 445* gives the two extensions their own algorithm object sets in place of `SupportedAlgorithms`: `SupportedAltPublicKeyAlgorithms` for the alternative key and `SupportedAltSignatureAlgorithms` for the alternative signature algorithm. Like their predecessors these sets are open, so they restrict nothing.

The corrigendum refers to the extensions as clauses 9.9.2 and 9.9.3 and once names `subjectAltPublicKeyInfo` where it means `altSignatureAlgorithm`. Since neither an earlier corrigendum nor the amendment rennumbers clause 9, we keep the 9.8 numbering of the 2019 edition.

Defect report 450. Defect report 450 disambiguates the term “signature component”, which the 2019 text of clause 7.2.2 uses for two different fields. The 2019 text first lists what a certificate may hold, among it “an alternative signature algorithm to be used instead of the algorithm specified in the signature component”. Here the signature component can only be the `signature` field of `TBSCertificate`, which carries the native algorithm identifier. Two sentences later, the CA’s rule for generating the alternative signature reads:

when generating the value in the `altSignatureValue` extension, exclude the signature component and the `altSignatureValue` extension from the public-key certificate, and generate the digital signature over the remaining DER encoded public-key certificate using the algorithm specified in the `altSignatureAlgorithm` extension

The same term is ambiguous here. It may again denote the `signature` field of `TBSCertificate`, or it may denote the `signature` component of the outer `SIGNED` structure, which holds the native signature value. The first meaning yields exactly the `PreTBSCertificate` that the Catalyst draft [TVF⁺18] had defined before the ITU-T text was written, the 2019 reading of Section 1.3: the native algorithm identifier is not signed and naturally also neither the outer `algorithmIdentifier` nor the outer `signature` (Figure 1a). Both existing implementations follow the draft (Section 6).

Corrigendum 3 resolves the ambiguity. It states that in the list of what a certificate may hold the term is to read “signature component of the `TBSCertificate` data type”, and that in the two generation rules “the ‘signature component’ should be followed by ‘of the `SIGNED` data type’”. The generation rule thus reads: exclude the signature component of the `SIGNED` data type, i.e. the outer signature value, and the `altSignatureValue` extension, and sign the remaining DER-encoded certificate. Under this Corrigendum 3 reading, the native algorithm identifier inside `TBSCertificate` is signed and, read literally, so is the outer `algorithmIdentifier` (Figure 1b). This is the meaning the 2019 text apparently intended.

The corrected text still leaves one point open, namely how the remainder of the certificate is encoded once the outer signature value and the `altSignatureValue` extension have been removed. The relying-party rule, which has the relying party “decode the public-key certificate and then re-DER-encode the same public-key certificate after the above modifications have been made”, suggests a re-encoded sequence of the two remaining components, the `TBSCertificate` and the outer `algorithmIdentifier`; the generation rule’s “remaining DER encoded public-key certificate” can equally be read as the bare concatenation of their encodings. The two encodings yield different inputs and thus incompatible signatures. We consider this encoding question unresolved by Corrigendum 3.

The clarification is not meant to introduce changes to the wire format or modify the processing rules. Nevertheless, it leads to interoperability problems with the existing FOSS implementations Bouncy Castle and wolfSSL that we analyze in this work. The two inputs differ in at least one DER element, the native algorithm identifier, so an alternative signature computed under one reading never verifies under the other. A verifier that follows the clarified text rejects every alternative signature issued so far by Bouncy Castle

Table 1: Normative statements of ITU-T X.509 (10/2019), as corrected by Corrigendum 3 (2026), on the MCA extensions.

Statement	Level	Clause
<code>altSignatureAlgorithm</code> and <code>altSignatureValue</code> are present together	shall	9.8.3, 7.2.2
<code>subjectAltPublicKeyInfo</code> is present independently of the two signature extensions (logical reading, see text)	may	9.8.2, 7.2.2
The alternative signature is created with the issuer’s alternative private key and verified with its alternative public key	shall	9.8.4
The alternative signature covers the DER encoding of the certificate without the outer signature value and without the <code>altSignatureValue</code> extension	shall	7.2.2 (Cor. 3)
The native signature covers the DER encoding including all three extensions	shall	7.2.2
A non-migrated relying party verifies the native signature	shall	7.2.2, 12.5.1
A migrated relying party verifies the alternative signature, after decoding, removing the excluded parts and re-encoding	shall	7.2.2, 12.5.1
A relying party supporting the extensions validates the alternative signatures of the certificates in a certification path that carry them	shall	7.7
The <code>SIGNED</code> type carries no second signature; its former <code>altAlgorithmIdentifier</code> and <code>altSignature</code> components are removed	shall	6.2 (Cor. 3)
The extensions are flagged critical or non-critical	may	9.8.2, 9.8.3
The extensions are flagged non-critical	recommended	9.8.2, 9.8.3 (notes)
The extensions are included in CA and end-entity certificates; the signature extensions also in CRLs	may	9.8.1
After the migration period, certificates are issued without the extensions	expected	7.2.2
The alternative algorithms are drawn from the object sets <code>SupportedAltPublicKeyAlgorithms</code> and <code>SupportedAltSignatureAlgorithms</code> , which are open	no restriction	9.8.2, 9.8.3 (Cor. 3)

or wolfSSL, and certificates issued under the clarified rule fail in both. Seen historically, it is the corrigendum that departs from the draft’s definition, which the implementations had followed.

None of the corrections adds a requirement that an alternative signature be present, and none touches clause 7.7 or 12.5.1. The downgrade of Section 2 therefore persists unchanged after all corrections.

1.5 Normative Requirements and Validation Paths

ITU-T X.509 is a common text with ISO/IEC 9594-8 and expresses provisions with the lowercase modal verbs “shall”, “should” and “may” of the ISO/IEC drafting rules [ISO21], which define them as requirement, recommendation and permission (clause 7, Tables 3 to 5); the specification itself states only that “shall” and similar obligatory language “are used to express requirements” [Int19]. These are the three categories that RFC 2119 [Bra97] names SHALL (equivalently MUST), SHOULD and MAY, and Table 1 uses them in that sense; RFC 2119 itself is not referenced by the specification. Recommendations given in notes are treated as SHOULD. Table 1 lists the normative statements of ITU-T X.509 on the MCA extensions.

Equally important is what the ITU-T specification does not say:

- It does not require a CA whose certificate carries an alternative public key to include alternative signatures in the certificates it issues. Issuing without the extensions remains permitted at any time.
- It does not give a unique recipe to determine whether a relying party should validate the native, the alternative, or both signatures. Instead, it implicitly assumes that the extension is used only with algorithms where a distinction between a “non-migrated” scenario algorithm and “migrated” scenario algorithm is given. This will not hold if both signatures are classified as PQC signatures.

The terms “non-migrated” and “migrated” relying party deserve a closer look, since they carry the entire decision logic of the mechanism. In effect, a multiple cryptographic algorithms certificate simply contains two signature algorithms, one in the native fields and one in the extensions. For any pair of algorithms encountered in the field, an implementation needs a decision rule that determines whether to validate the native signature, the alternative signature, or both. The ITU-T specification veils this necessity behind the implicit assumption that there is always a clear old-versus-new relationship between the two algorithms, so that a relying party is either “not yet” or “already” at the alternative one and, once it supports the alternative algorithm, prefers it unconditionally. This is a conceptual veil that adds no technical clarification. It is not even clear that an implementer should always prefer the alternative signature over the native one: a reasonable security policy based on the concrete algorithms and parameters may well decide differently, for instance when the alternative scheme or its parameter set is less trusted than the native one, especially when the native algorithm is itself a PQC algorithm.

Figures 2 and 3 illustrate the resulting validation paths in a hierarchy of a root CA, a sub-CA and an end entity in which every certificate carries the extensions. Black arrows denote certificate issuance; gray arrows connect the public key used for verification to the signature it verifies. Figure 2 shows the two paths the ITU-T specification defines, side by side: a non-migrated relying party verifies only the native signatures (a), a migrated one only the alternative signatures (b). In neither case are both signatures of a certificate verified. Figure 3a shows a mixed path in which the sub-CA certificate is verified with its native signature and the end-entity certificate with its alternative signature. A migrated relying party takes this path if it cannot validate the alternative signature algorithm given in the `altSignatureAlgorithm` extension, and thus the `altSignatureValue`, of the sub-CA certificate. The ITU-T specification neither forbids nor mentions this case. Figure 3b shows a relying party that verifies both signatures of every certificate, a stricter policy the ITU-T specification does not mandate. Section 2 shows that none of these policies protects a relying party against a quantum attacker, because a certificate without the extensions is always acceptable.

1.6 Related Work

The downgrade at the centre of Section 2 was foreseen by the authors of the Catalyst draft [TVF⁺18]. Its security considerations state that “a quantum-enabled adversary could create a certificate or CRL without an alternative signature, and forge the conventional signature of any issuer, causing upgraded applications to accept forged credentials”, and its verification rules prescribe the corresponding guard: if the issuer holds an alternative public key but the subject certificate carries no alternative signature, “the verifier SHOULD reject the subject certificate”. The draft’s closing note further concedes that it leaves “how to signal that multiple signatures should have been present” to the protocols and applications. ITU-T X.509 adopted the extensions but not the guard, and Section 2 is, in that light, an account of what follows from that omission. It gives a precise attack model with a

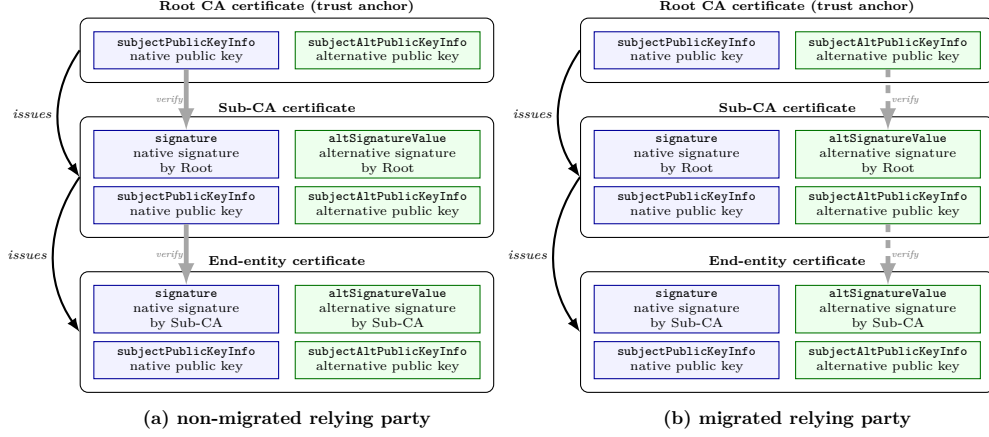


Figure 2: The two validation paths defined by ITU-T X.509, shown separately: (a) a non-migrated relying party verifies only the native signatures, (b) a migrated relying party verifies only the alternative signatures. In neither case are both signatures of a certificate verified.

sub-CA and an end-entity variant, to which Section 6 provides the confirmation against a real FOSS implementation. The draft’s final version of 2023, which set the work aside, describes the ITU-T variant as one that “allows two keys to be placed in a certificate but only one used at a time” [TVF⁺23], that is, in the same one-key-at-a-time terms as the migrated/non-migrated framing analysed in Section 1.5.

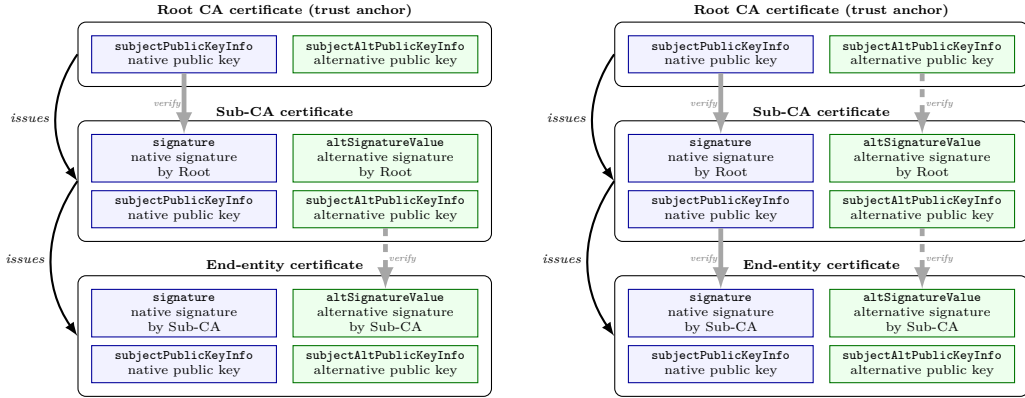
Two recent empirical studies test hybrid certificate formats against deployed verifiers. Lee et al. [LSE⁺26] compare Catalyst, composite and chameleon certificates across eleven verifiers and report, for Catalyst, that Bouncy Castle checks the alternative signature only through an opt-in call, that wolfSSL checks a present alternative signature but cannot require one, and that a certificate without its alternative signature is accepted by every verifier; they conclude that the format should be set aside on security grounds before any cost ranking. Kim et al. [KCKK26] examine eight path validation implementations and find that they do not support the MCA extensions, observe that no interface lets a deployment declare the “migrated” state of the ITU-T text or report whether it acted on it, and propose a policy parameter for verifier interfaces. Both studies correctly identify the possibility of the issuance of forged native-signature-only certificates. However, neither analyses the open questions regarding the specification’s decision rule or considers the RFC 5280 validity of a certificate accepted on its alternative signature alone.

2 Lack of Opt-In to PQC-enabled CAs or Sites

2.1 No Opt-In Signal in the Extensions

Section 1 describes the MCA extensions and their verification rules. As detailed there, a migrated relying party verifies the alternative signature – *if the certificate carries one* with an algorithm it can validate – and, in the absence of any other decision rule in the ITU-T specification (Section 1.5), verifies the native signature otherwise. The weakness that follows was foreseen by the Catalyst draft, whose SHOULD-level guard of requiring alternative signatures from CAs that have an alternative key the ITU-T specification did not adopt (Section 1.6). This leads to the vulnerability to a quantum attacker that is identified in previous work. It is unaffected by the corrigenda to the specification (Section 1.4).

The crucial point is the conditional verification of the alternative signature. Neither



(a) A mixed validation path: the sub-CA certificate is verified with its native signature, the end-entity certificate with its alternative signature. A migrated relying party takes this path if it cannot validate the alternative signature algorithm given in the `altSignatureAlgorithm` extension, and thus the `altSignatureValue`, of the sub-CA certificate.

(b) A strict validation policy that verifies both signatures of every certificate. The ITU-T specification does not mandate it.

Figure 3: Two validation policies that the ITU-T specification neither defines nor forbids: (a) a mixed path, (b) both signatures verified on every certificate.

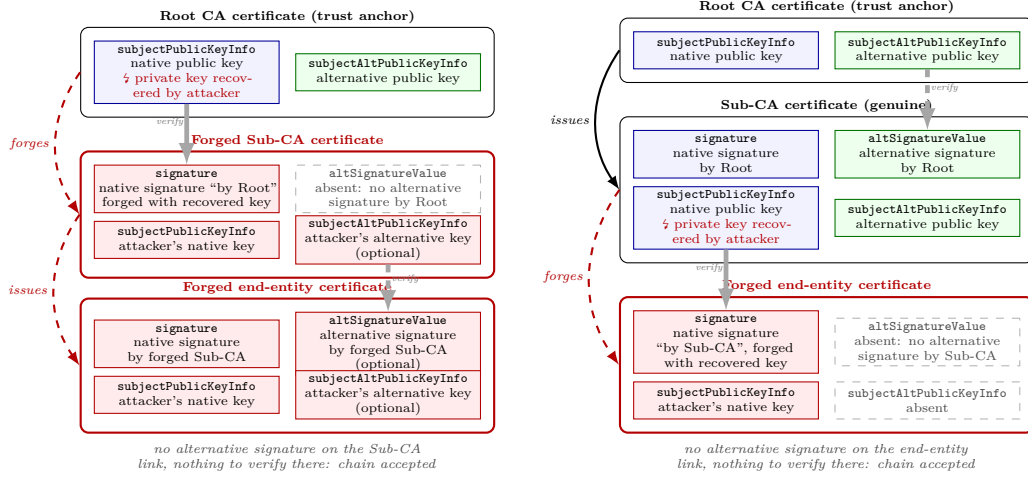
the certificate profile nor the path validation algorithm contain a rule of the form “if the issuer holds an alternative key, then every certificate it issues must carry an alternative signature”. A CA is free to issue certificates without the MCA extensions even though its own certificate carries an alternative key, for instance for subscribers or sub-CAs that have not yet migrated. Consequently, when a relying party encounters a certificate that carries only a traditional signature it has to accept it according to the ITU-T specification. Neither the trust anchor, nor an intermediate CA, nor an end entity can opt in to PQC verification in a way that a relying party could enforce. In the following we show in detail how this allows a quantum attacker to bypass the PQC layer of such a hierarchy entirely.

2.2 Attacker Model

We consider a quantum attacker, i.e. an adversary who can recover the private key belonging to any traditional public key of their choice (RSA, ECDSA, EdDSA), but cannot break the PQC signature scheme used for the alternative keys. The attacker is a network attacker in the protocol that processes the certificates, e.g. TLS or S/MIME, and can therefore present arbitrary certificate chains to the relying party. The relying party has migrated in the sense of Section 1.3: it trusts the root CA including its alternative key, verifies the alternative signature of every certificate that carries one with an algorithm it supports, and verifies the native signature otherwise. Whether it additionally verifies the native signature of certificates that carry both, as in Figure 3b, is irrelevant for what follows. Figure 2 (b) shows the intended situation, where each link of the chain is protected by a traditional and an alternative PQC signature and the relying party verifies the alternative one.

2.3 Attack 1: Downgrade of a Sub-CA

In the first variant, shown in Figure 4a, the attacker recovers the traditional private key of the root CA. With it, the attacker creates a forged sub-CA certificate that contains an arbitrary subject name, for instance that of the genuine sub-CA, the attacker’s own



(a) Quantum Attack 1, scenario QAT1 in Table 2. The attacker has recovered the root's traditional private key (⚡) and forges a sub-CA certificate that carries a valid native signature under that key but no alternative signature by the root (dashed slot), with the attacker's own keys; red boxes and fields are created by the attacker. The relying party can verify only the native signature on the sub-CA link, finds no alternative signature there, and accepts the chain. Below the forged sub-CA even alternative signatures verify, against the attacker's own alternative key. Compare the intended path in Figure 2 (b).

(b) Quantum Attack 2, scenario QAT2 in Table 2. Root and sub-CA are genuine and the sub-CA link is verified through its alternative signature. The attacker has recovered only the sub-CA's traditional private key (⚡) and forges an end-entity certificate with a valid native signature under it and no alternative signature (dashed slots). The relying party verifies the native signature on the last link, finds no alternative signature there, and accepts the chain. Compare the intended path in Figure 2 (b).

Figure 4: The two quantum attacks of Section 2, scenarios QAT1 and QAT2 in Table 2: (a) forged sub-CA, (b) forged end-entity certificate.

traditional or PQC public key as the subject key, and no alternative signature by the root, that is neither an `altSignatureValue` nor an `altSignatureAlgorithm` extension. It may nonetheless carry the attacker's own `subjectAltPublicKeyInfo`, since the subject's alternative key is the issuer's choice, so that the end-entity certificates issued below it can bear valid alternative signatures. The certificate is signed with the recovered traditional root key only. Below this forged sub-CA the attacker then issues arbitrary end-entity certificates, with either traditional or PQC signatures.

For the relying party, path validation succeeds: the traditional signature of the root over the forged sub-CA certificate verifies, and since the forged certificate carries no alternative signature, there is none to verify. The forged end-entity certificate below it verifies with whatever signature the attacker chose to put on it. Nothing in the chain and nothing in the trust anchor configuration tells the relying party that an alternative signature should have been present. The genuine sub-CA certificate, which the root did sign with its alternative key, is never involved; the attacker simply presents the forged one in its place. The attacker does not even need to know that the genuine sub-CA exists.

2.4 Attack 2: Downgrade of an End-Entity Certificate

In the second variant, shown in Figure 4b, the genuine sub-CA certificate remains in the chain. It carries an alternative key and an alternative signature by the root, and both verify. The attacker recovers only the traditional private key of the sub-CA and uses it to issue a forged end-entity certificate: a server certificate for the site the attacker wants to impersonate, or a client certificate in the name of a user the attacker wants to impersonate.

The forged certificate contains the attacker’s key and only a traditional signature by the sub-CA.

The migrated relying party verifies the link from the root to the sub-CA with the alternative signature, verifies the traditional signature of the sub-CA over the forged end-entity certificate, finds no alternative signature on it, and accepts the chain. This variant is notable because the PQC layer *is* exercised in the upper part of the chain and still provides no protection: the attacker chooses the level at which the downgrade happens, and the lowest level suffices.

2.5 Consequences

Both attacks result in a complete authentication bypass in the protocol that uses the certificates. Depending on which end-entity certificate is forged, the attacker impersonates a server, i.e. a site, towards a client, or a client, i.e. a user, towards a server; the attack applies to TLS server and client authentication as well as to S/MIME signatures and encryption in the same way. The cost for the attacker is the recovery of exactly one traditional private key, that of the root or that of the sub-CA, which is the same cost as against a purely traditional hierarchy. Adding the MCA extensions to a CA hierarchy therefore provides no security against a quantum attacker unless the relying party is told, by some means outside the certificates, that it must refuse a certification path in which any link lacks an authentic PQC signature from the issuer.

2.6 Migration States and Policies

Before turning to countermeasures we make precise what “migrated” can mean for a relying party, since the countermeasure depends on it. We adopt a deliberate simplification throughout: post-quantum algorithms appear only in the MCA extensions, never as native algorithms. This is the stated intention of the mechanism and its probable use, and it lets us call the alternative signature of a link its post-quantum signature.

A relying party is in one of two migration states. A *non-migrated* relying party does not process the extensions, i.e. it validates according to RFC 5280 alone, under which the unknown non-critical extensions are ignored. It verifies the native signature of every certificate, interoperates with every issuer, migrated or not, and obtains no security against the quantum attacker. A *migrated* relying party processes the extensions and supports some set of post-quantum algorithms. That set is not necessarily complete with respect to the possible signature algorithms encountered in the field: new algorithms and parameter sets are specified and deployed while the implementation is in the field, so a migrated relying party must expect to meet issuers whose alternative key it cannot process. What it does then, and what it does on a link whose issuer has no alternative key at all, is a matter of policy. We distinguish three policies, illustrated in Figure 5.

- *Opportunistic-PQC*. On a link whose issuer carries an alternative key of a supported algorithm, the relying party requires and verifies the alternative signature. On a link whose issuer carries an alternative key of an unsupported algorithm, and on a link whose issuer carries no alternative key, it verifies the native signature. It interoperates with every issuer. It has post-quantum security on exactly the links whose issuer algorithm it supports. On the other links the attacker of Section 2.2 succeeds as in Attack 2 with the issuer’s recovered native key.
- *Require-PQC-from-capable-issuers*. As Opportunistic-PQC, except that a link whose issuer carries an alternative key of an unsupported algorithm is rejected. This is the guard of the Catalyst draft (Section 1.6) made sound. It has post-quantum security on every link whose issuer has migrated and none on the links whose issuer has not. Its price is that it must support every algorithm that any capable issuer in the path

uses. Since that set is in flux, legitimate chains become unverifiable for it whenever an issuer adopts an algorithm it lacks.

- *Require-PQC*. Every link must carry an alternative signature that the relying party can verify. A chain in which any issuer has no alternative key is rejected. This is stage 3 of Section 1.1, mandatory PQC, for MCA chains. It has post-quantum security on every link and interoperates only with fully migrated hierarchies.

All three policies take the presence of an alternative key in an issuer certificate as an implicit signal that alternative signatures are to be expected below it. Under all three policies two conditions are fatal: an alternative signature that is absent although the issuer carries an alternative key the relying party supports, and an alternative signature algorithm that is inadmissible for the issuer’s key or unsupported. Both sit in the subject certificate, which is the one the attacker forges. The issuer’s key algorithm, by contrast, sits in the issuer certificate, which is genuine by the time it is consulted: it is either the trust anchor or has been validated one link higher under the same policy. The native fallbacks of the first two policies therefore concern genuine issuers only, and each such fallback leaves its link without post-quantum security by the relying party’s own choice.

The rule of the ITU-T specification – verify the alternative signature if present, never require it – is none of the three. Figure 5 lists it as the policy ITU-T-2019 beside them. It is stage 1 of Section 1.1, and its outcome against the quantum attacker is that of a non-migrated relying party (Section 2.1). For a relying party under Require-PQC the MCA chain is a post-quantum chain: the native signatures are dead weight to it, and the mechanism’s remaining benefit is that one certificate serves the migrated and the non-migrated population in place of two parallel chains.

2.7 The Countermeasures in Detail

This section first states the countermeasure of Section 2.6 precisely, as an algorithm, and then widens the view to opt-in signals outside the MCA extensions. The most direct approach to achieve security against a quantum attacker, which the Catalyst draft proposed, is to require an alternative signature on the certificate whenever the issuer has an alternative key. However, if we look closely, we find that even an implementation that fulfills this requirement might be vulnerable to the downgrade. The reason is that the verification of the alternative signature is still conditional, namely, it depends on whether the relying party can verify it. Since in the case at hand the entire certificate is forged, the choice of the `altSignatureAlgorithm` in it is at the discretion of the attacker. While being unable to produce a valid alternative signature verifiable against the issuer certificate, the attacker can choose an algorithm identifier that is not supported by the relying party. The sound specification of the countermeasure is therefore more involved than the guard alone. Algorithm 1 states this, parameterised by the policy of Section 2.6; Appendix B shows its control flow as a flowchart. Its lines 20 and 23 close the gap in this guard: a signature algorithm that the relying party cannot verify, whether because it is not admissible for the issuer’s key or merely unsupported, must fail the validation, since the relying party cannot tell an attacker’s choice from a legitimate issuer’s choice.

For the post-quantum signature algorithms standardised so far, ML-DSA, SLH-DSA and the stateful hash-based schemes XMSS and LMS including their multi-tree variants XMSS^{MT} and HSS, each key algorithm admits exactly one signature algorithm, so line 20 reduces to an equality check and line 23 to the support of A_K itself. For the stateful schemes the support check may extend to the parameter set, which is carried inside the signature rather than in the algorithm identifier (Section 5), or, alternatively, an unsupported parameter set may simply lead to a validation failure. Both achieve the stated security goal. A relying party that admits alternative key algorithms that can sign with different signature algorithms has to maintain the admissibility relation of line 20 explicitly.

Algorithm 1 Processing of the alternative signature of one certificate under the require-alternative-signature policies of Section 2.6: OPP (Opportunistic-PQC), REQ-CAP (Require-PQC-from-capable-issuers) and REQ (Require-PQC). The native signature is otherwise processed under the relying party's usual rules; the algorithm only decides whether the alternative signature satisfies the policy. $\mathcal{S}$ denotes the set of pairs (key algorithm, signature algorithm) that the relying party can verify.

Input: subject certificate C , issuer certificate I , supported pairs $\mathcal{S}$, policy $\in \{\text{OPP}, \text{REQ-CAP}, \text{REQ}\}$

Output: VALID or INVALID

```

1: if  $I$  carries no subjectAltPublicKeyInfo extension then
2:   if policy = REQ then
3:     return INVALID ▷ unmigrated issuer not accepted
4:   else
5:     return the result of validating the native signature of  $C$  against
        $I.\text{subjectPublicKeyInfo}$  ▷ no post-quantum security on this link
6:   end if
7: end if
8:  $(A_K, K_{\text{alt}}) \leftarrow$  algorithm and subjectAltPublicKey of  $I.\text{subjectAltPublicKeyInfo}$ 
9: if  $A_K$  is unknown to the relying party then
10:   if policy = OPP then
11:     return the result of validating the native signature of  $C$  against
         $I.\text{subjectPublicKeyInfo}$  ▷ no post-quantum security on this link
12:   else
13:     return INVALID ▷ issuer algorithm unsupported
14:   end if
15: end if
16: if  $C$  carries no altSignatureAlgorithm or no altSignatureValue extension then
17:   return INVALID ▷ the Catalyst draft guard proper
18: end if
19:  $A_S \leftarrow C.\text{altSignatureAlgorithm}$ 
20: if  $A_S$  is not admissible for keys of algorithm  $A_K$  then
21:   return INVALID ▷ attacker-chosen or malformed identifier
22: end if
23: if  $(A_K, A_S) \notin \mathcal{S}$  then
24:   return INVALID ▷ unsupported: no fallback to the native signature
25: end if
26:  $m \leftarrow$  input of the alternative signature under the coverage rule in force (Section 1.3)
27: if  $\forall \text{fy}_{A_S}(K_{\text{alt}}, m, C.\text{altSignatureValue})$  fails then
28:   return INVALID
29: end if
30: return VALID

```

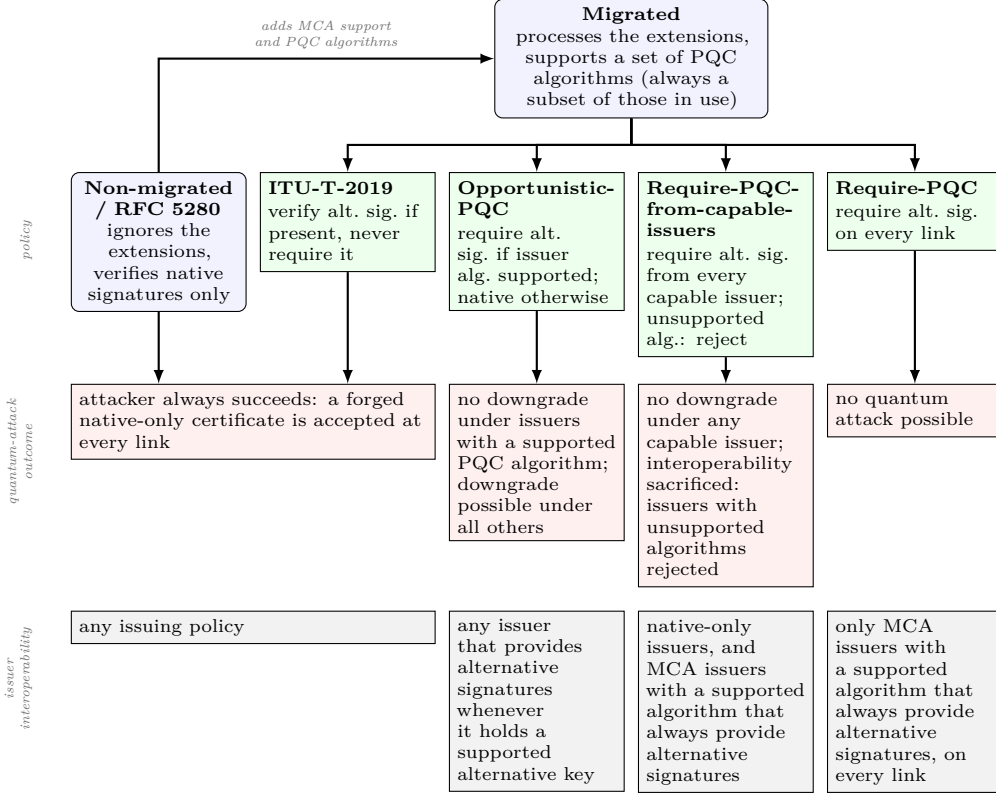


Figure 5: Migration states of a relying party, the policies of the migrated state, and the outcome against the quantum attacker for each. The two states differ in whether the extensions are processed. ITU-T-2019 is the rule of the specification itself. The three other policies differ in how a link is treated whose issuer has no alternative key or an alternative key of an unsupported algorithm. The outcome of ITU-T-2019 is that of a non-migrated relying party. The bottom row names the issuing policies each relying party interoperates with.

The policy then rejects every certificate whose issuer chose a signature algorithm that is admissible for its key but not supported by the relying party.

Lines 1 and 9 are the policy branches. They concern the issuer certificate, which is genuine when the algorithm reaches it, so neither branch is open to the attacker; but each native fallback leaves its link without post-quantum security, and there Attack 2 succeeds with the issuer’s recovered native key. Which fallbacks a relying party accepts is exactly what its policy decides (Section 2.6).

The absence of an opt-in signal for a commitment to PQC signatures, which we described as part of stage 2 of the PQC migration in Section 1.1, could be resolved in several ways, each requiring additional specifications and requirements for relying parties as well. A non-critical policy extension in all issuer certificates in the path, including the root CA certificate that serves as the trust anchor, could tell a client that understands it that the issuer will always provide a post-quantum signature – as the native signature or in the alternative one, either pinned to one of the two or left open. A caveat applies: under RFC 5280 [Coo] the contents of the trust anchor’s to-be-signed part do not enter certification-path validation, so such an extension in the trust anchor would strictly be ignored. However, honouring extensions only below the trust anchor would make the application of the MCA mechanism impossible in general and thus here we use the assumption that trust anchors are evaluated just as other CA certificates in the path according to RFC 5280.

The same signal for a full-PQC chain could instead be placed in the end-entity certificate, opting in a specific end entity such as a website or an email account rather than the whole chain. Outside the certificates, a per-protocol mechanism in the spirit of HTTP Strict Transport Security is possible. A concrete example for TLS is the PQC Continuity draft [SR26]: when a server presents a post-quantum certificate, it includes a `pq_cert_available` extension in the first `CertificateEntry`, carrying an `algorithm_validity_period` in seconds. The client caches this commitment per server identity, port and protocol and, within the period, refuses traditional-only certificates and offers only PQC-capable signature algorithms. This is HTTP Strict Transport Security (HSTS) [HJB12] moved to the TLS layer. It needs no change to the CA infrastructure, but like HSTS it protects only after a first successful post-quantum connection unless the commitment is preloaded. The drawback of such mechanisms is that they must be defined separately for each protocol. Finally, the opt-in can be kept entirely out of band in a preload list of PQC-committed origins shipped with the client, as in the Chrome proposal (Section 1.1). The design space thus ranges from the implicit signal given by the presence of the MCA extensions under a corresponding issuing policy (Section 2.6), through certificate extensions with chain-wide or per-end-entity scope and per-protocol signalling, to out-of-band preload lists.

3 The Alternative Signature as a Stand-alone Signed Object: Formal EUF-CMA Violations

Under both the 2019 reading and the Corrigendum 3 reading of the signature coverage the alternative signature is an ordinary signature, under the CA’s alternative key, over a byte string that is itself the DER encoding of a structured value. This section shows that anyone who holds an MCA certificate can re-package that signature as a new signed object of the generic shape defined in clause 6.2 – a to-be-signed part, an algorithm identifier and a signature over the to-be-signed part – whose to-be-signed part the CA never signed as such. We first fix the formal setting, then give the transformation under both readings, and finally assess what it means.

3.1 The Signed-Object Scheme of X.509

A signature scheme $\Sigma = (\text{KGen}, \text{Sign}, \text{Vfy})$ is existentially unforgeable under chosen-message attack (EUF-CMA) [GMR88] if no efficient adversary that may obtain signatures $\text{Sign}(\text{SK}, m)$ on messages m of its choice can output a pair (m^*, σ^*) with $\text{Vfy}(\text{PK}, m^*, \sigma^*) = 1$ for a message m^* it never submitted. The messages are arbitrary byte strings. What a byte string *means* is outside the definition.

X.509 gives the byte strings a meaning through the parameterised `SIGNED` type of clause 6.2 (Listing 1). In the X.509 context, signing typically produces a value of the form `SEQUENCE{t, alg, Sign(SK, DER(t))}`, where *alg* identifies Σ . A verifier checks the signature over `DER(t)`, which for a correctly encoded object is simply the octet string received as the `toBeSigned` component.

This is the scheme under which every X.509 certificate, CRL and attribute certificate is signed, and it is the scheme under which a CA’s alternative key is meant to be used once the migration period is over and the new algorithms move into the base certificate (clause 7.2.2, Section 1.3). While the ITU-T specification does not explicitly suggest the re-certification of an alternative CA key as a native key, it also does not prohibit it. We therefore consider the joint model in which one key pair $(\text{PK}_{\text{alt}}, \text{SK}_{\text{alt}})$ is used under both signing rules of the standard: the rule of clause 6.2 for stand-alone signed objects and the rule of clauses 7.2.2 and 9.8 for alternative signatures. The two rules sign different inputs derived from the same certificate, and nothing separates their domains: no prefix,

```

-- MCA certificate C
SEQUENCE {
  toBeSigned SEQUENCE {
    version v3,
    serialNumber n,
    signature alg-nat,
    issuer ...,
    validity ...,
    subject ...,
    subjectPublicKeyInfo pk-nat,
    extensions {
      ...,
      subjectAltPublicKeyInfo pk-alt,
      altSignatureAlgorithm alg-alt,
      altSignatureValue sig-alt
    },
    algorithmIdentifier alg-nat,
    signature sig-nat }
}

-- stand-alone object O(C)
SEQUENCE {
  toBeSigned SEQUENCE {
    version v3,
    serialNumber n,
    -- (signature omitted)
    issuer ...,
    validity ...,
    subject ...,
    subjectPublicKeyInfo pk-nat,
    extensions {
      ...,
      subjectAltPublicKeyInfo pk-alt,
      altSignatureAlgorithm alg-alt,
      -- (altSignatureValue omitted)
    },
    algorithmIdentifier alg-alt,
    signature sig-alt }
}

```

Listing 2: The transformation of Section 3.2 in abbreviated ASN.1 value notation. Left: the MCA certificate C . Right: the stand-alone signed object $O(C)$ under the 2019 reading. Under the Corrigendum 3 reading (Section 3.3) the `toBeSigned` part of the stand-alone object is instead, if the remainder is re-encoded as a sequence (Section 1.4), the sequence of the `TBSCertificate` without `altSignatureValue` (the `signature` field retained) and the native `algorithmIdentifier`.

no wrapper type and no context string distinguishes an alternative-signature input from a to-be-signed value.

3.2 Transformation under the 2019 Reading

Let C be an MCA certificate whose alternative signature was computed under the 2019 reading of the signature coverage, i.e. over the draft's `PreTBSCertificate`, as Bouncy Castle and wolfSSL do (Section 6). The input of the alternative signature is $\text{DER}(\text{PreTBS}(C))$, where $\text{PreTBS}(C)$ is the `TBSCertificate` of C without its `signature` field and without the `altSignatureValue` extension (Figure 1a). $\text{PreTBS}(C)$ is an ASN.1 value in its own right. Form

$$O(C) = \text{SEQUENCE}\{ \text{PreTBS}(C), \text{altSigAlg}(C), \text{altSigValue}(C) \},$$

where $\text{altSigAlg}(C)$ and $\text{altSigValue}(C)$ denote the values of the `altSignatureAlgorithm` and `altSignatureValue` extensions of C , i.e., take the alternative algorithm identifier and the alternative signature value out of their extensions and place them where a certificate carries its algorithm identifier and its signature (Listing 2). $O(C)$ is a syntactically valid `SIGNED` value, and the generic verification of clause 6.2 accepts it under PK_{alt} : it takes the `toBeSigned` component, which is exactly $\text{DER}(\text{PreTBS}(C))$, and verifies the alternative signature over it, which is what the CA computed. Thus, $O(C)$ resembles a certificate signed with the alternative algorithm alone, lacking only the `signature` field of its to-be-signed part in relation to a `TBSCertificate`.

In the joint model of Section 3.1 this is an existential forgery against the signed-object scheme of clause 6.2 keyed with PK_{alt} . The CA never signed the value $\text{PreTBS}(C)$ as a to-be-signed value; it signed the certificate C , whose to-be-signed value is a different one. The adversary needs no signing query of its own: any subscriber of the CA obtains C by requesting a certificate, with content of its choosing, and anyone who sees C can carry out the transformation.

3.3 Transformation under the Corrigendum 3 Reading

Under the corrected text (Section 1.4) the situation is similar, only that the forged object has a different structure, which does not anymore match the pattern of the `SIGNED` type.

Nevertheless, it is also a sequence of data never signed by the key holder followed by a valid signature over that data. The concrete structure of the data depends on the exact reading of Corrigendum 3 with regard to the scope of the signed data, which we consider still as ambiguous, as stated in Section 1.4. We therefore omit the detailed analysis of this variant.

3.4 Assessment

What formally fails here is the abstraction that the standard itself defines, the signed object of clause 6.2, and it fails because the standard uses one key under two signing rules without separating their domains. The violation is formally manifest, but it leads to an exploitable vulnerability only if some verifier accepts the transformed to-be-signed value as a meaningful object under the CA’s alternative key. Two conditions must hold for that.

First, the verifier must know PK_{alt} as a key that signs stand-alone objects. In an MCA hierarchy the issuer’s alternative public key exists only inside the `subjectAltPublicKeyInfo` extension of the issuer’s certificate, and the generic verification of a `SIGNED` object never looks for a key there. The transformed object is therefore verifiable in practice only once the same alternative key is also certified as the *native* key of a stand-alone certificate – a PQC-only CA certificate issued in parallel, or the CA’s certificate after the migration period.

Second, the transformed to-be-signed value must decode as something the verifier acts on. An MCA certificate is a version-3 certificate, so $PreTBS(C)$ begins with the explicitly tagged version, followed by the serial number and then the issuer name in the position where a `TBSCertificate` requires the `signature` algorithm identifier; a `Name`, even an empty one, never decodes as an `AlgorithmIdentifier`, whose first component is mandatory and an object identifier. A `TBSCertList` begins with an optional integer version and then the algorithm identifier, an `AttributeCertificateInfo` with an integer version, and an `OCSP ResponseData` with the tagged version followed by a tagged responder identifier; each fails on its first or second component. Hence no conforming X.509 or OCSP verifier accepts $O(C)$ as a certificate, CRL, attribute certificate or OCSP response, and we know of no practical exploitation.

This guarantee, however, is an accident of the encodings, not a property the specification establishes, and it has to be re-verified for every further context in which the alternative key is used: every other to-be-signed type of X.509, CMS signed attributes, and potentially asynchronous protocols.

A design that separated the domains would have mitigated or avoided the formal problem altogether: either a dedicated wrapper type for the alternative-signature input, for instance a sequence whose first component is a distinguished object identifier, or a context string of the kind that ML-DSA and SLH-DSA provide for exactly this purpose [Nat24]. However, a generic mechanism defined for arbitrary algorithms cannot rely on the latter. The simplest mechanism to reliably prevent the formal EUF-CMA violations is to not allow the re-certification of a key that is or was in use as an alternative signature key.

4 Partial Verification leads to Malleable Certificates and Arbitrary Content Injection

Somewhat related to the formal EUF-CMA violations discussed in Section 3 is the malleability of the native signature value, and, in the 2019 reading of the signature coverage, the native signature algorithm identifier of a certificate.

A certificate that a relying party accepts on the basis of its alternative signature may nonetheless be invalid according to RFC 5280 [Coo]. Two properties combine to cause this: the native signature is not covered by the alternative signature (Section 1.3), and the

native signature may not be verified at all when the alternative one is (Section 5). An application that receives the validated path from the path-validation routine may therefore be handed a path that is invalid under RFC 5280 and mishandle it in later processing.

This gap lets an attacker replace the native signature value at will and, for the two existing implementations, also the native algorithm identifiers: they compute the alternative signature over the draft's `PreTBSCertificate` (the 2019 reading), which omits the `signature` field (Section 6), whereas under Corrigendum 3 both algorithm identifiers are covered (Section 1.4). Manipulating the identifiers can influence the decision logic of the mechanism itself. Either way the attacker gains no advantage in forging certificates: an attacker who could downgrade the native algorithm to one they can forge would already be able to produce a forged certificate without the downgrade considered here. The risk lies elsewhere. Once the validated chain is returned to the application and is there naturally believed to be authentic, a non-quantum network attacker can achieve the following:

- The attacker can use the unverified native signature value – and, in the existing implementations, the native algorithm identifiers – as a container for arbitrary data of arbitrary size. This might be leveraged in denial of service attacks or as the first step of an exploit that smuggles malicious binary data into the target system.
- In the existing implementations a secondary cryptographic-policy check applied to the returned chain may run on the attacker-manipulated native signature-algorithm identifier and produce a wrong result, such as wrongly accepting the chain.
- The attacker might even modify the signature such that it is a valid signature under an issuer certificate crafted by them. While it is not apparent how this can be leveraged to a generic attack, it seems to bear potential for confusion. For instance, the application might be returned the peer's validated EE certificate with the crafted native signature which chains it correctly to an untrusted CA certificate crafted by the attacker. (It makes no sense to assume that the attacker can craft certificates that are trusted, as in the typical PKI model this allows general and arbitrary certificate forgeries and thus gives them an even stronger position.)

We point out that the only widely used FOSS implementation that supports the MCA certificates in its path validation routine, `wolfSSL`, adopted the stricter verification model where the native signature is always verified, thus retaining RFC 5280 compliance and preventing the malleability problems pointed out in this section.

5 Unspecified Verifier Behaviour

In this section we point out various details where the ITU-T specification leaves verifier behaviour unspecified.

First, it does not address the case in which the algorithm identifier of the alternative signature is recognised as supported but its concrete parameters are not. This is not hypothetical: for the stateful hash-based schemes LMS and XMSS the parameter set is carried inside the signature itself, so a verifier can recognise the algorithm and still be unable to process the particular instance.

Second, it leaves open whether path validation also verifies the native signature: (a) always, or (b) only when it cannot verify the alternative one. As reported in Section 6, `wolfSSL` follows (a) and always verifies the native signature, and conversely does not fall back to the alternative signature when the native one cannot be verified (S10 to S13 in Table 2).

Third, even the Corrigendum 3 reading of the signature coverage leaves open how the input of the alternative signature is encoded, as a re-encoded sequence of the remaining components or as their bare concatenation (Section 1.4). Two implementations that both

follow the corrigendum may therefore still produce and expect incompatible alternative signatures.

Finally, the ITU-T specification neither defines nor restricts the set of algorithms that may appear in the extensions. The apparently intended split – a post-quantum alternative algorithm and a traditional native one – therefore need not hold. The mechanism could equally carry a hybrid of two different post-quantum schemes. An implementer confronted with an arbitrary pair of algorithms may then adopt a complex, policy-dependent strategy for deciding which signature to validate. Different implementations may resolve the same certification path differently, which leads directly to interoperability problems even between implementations that support the MCA mechanism.

6 Implementations

At least two libraries implement the MCA extensions: Bouncy Castle² and wolfSSL³. We determined their features with a proof-of-concept test tool that generates the test chains with Bouncy Castle and validates them with both libraries (Table 2). Each chain is a root CA, a sub-CA and an end-entity certificate, $\text{Root} \rightarrow \text{Sub-CA} \rightarrow \text{EE}$.⁴ In wolfSSL the root is loaded as the trust anchor, the sub-CA certificate is verified against it and then added as a CA, and the end-entity certificate is verified last.

6.1 Bouncy Castle

Bouncy Castle [Bou] can both create and verify the extensions. A certificate is created with `X509v3CertificateBuilder.build(signer, isCritical, altSigner)`, which computes the alternative signature with the `altSigner` over the draft’s `PreTBSCertificate`, i.e. the `TBSCertificate` with its `signature` field and the `altSignatureValue` extension removed (the 2019 reading) (`setSignature(null)` followed by `generatePreTBSCertificate()`), adds the `altSignatureAlgorithm` and `altSignatureValue` extensions, and then computes the native signature with the `signer`. Verification is offered as two per-certificate primitives on `X509CertificateHolder`: `isSignatureValid` for the native signature and `isAlternativeSignatureValid` for the alternative one. The latter reconstructs the same `PreTBSCertificate`. Bouncy Castle therefore deviates from the input as clarified by Corrigendum 3 (Section 1.4). Crucially, the mechanism is not integrated into path validation. We confirmed that Bouncy Castle’s PKIX `CertPathValidator` verifies the native signature only and ignores the alternative one: a chain whose alternative signature is invalid but whose native signature is valid is accepted. When the extensions are marked critical, the same validator rejects the chain, treating them as unknown critical extensions. The alternative signature can therefore be checked only by the application, explicitly and per certificate. The application must itself locate the issuer’s alternative public key in the issuer certificate’s `subjectAltPublicKeyInfo` extension and decide which signatures to verify. Since the test results presented below are referring to the certification path validation, Bouncy Castle has to be considered as an implementation that does not support the MCA certificates for our purpose.

6.2 wolfSSL

wolfSSL [wol24] implements the same extensions under the name dual-algorithm or Chimera certificates, enabled by the build option `WOLFSSL_DUAL_ALG_CERTS`. In contrast

²<https://www.bouncycastle.org>

³<https://www.wolfssl.com>

⁴The chains use a native RSA-3072 and an alternative ML-DSA-65 key and signature. The proof-of-concept and its results accompany this paper.

Table 2: Path-validation results for the fifteen test chains (native RSA-3072, alternative ML-DSA-65); each is a root CA, a sub-CA and an end-entity certificate (Root $\rightarrow$ Sub-CA $\rightarrow$ EE). Each entry is the verdict of Bouncy Castle’s **PKIX CertPathValidator** and of wolfSSL’s **CertManager** on the chain; in wolfSSL the sub-CA certificate is verified against the root before it is added as a CA. Bouncy Castle verifies the native signature only; the invalid alternative signature of S2 is thus accepted, and the critical extensions of S4 are rejected as unknown. wolfSSL verifies the alternative signature (S2 rejected). S5 to S9 probe the strict reading of clause 7.2.2 (Section 1.3): wolfSSL accepts an alternative key without an alternative signature (S5, S7) and an alternative signature without an alternative key (S6), and verifies the alternative signature in those shapes (S8, S9 rejected). S10 to S13 give a certificate a native signature that wolfSSL cannot verify, because the algorithm is unknown to it or the signature is invalid, together with a valid alternative signature; wolfSSL rejects all of them, so it never falls back to the alternative signature. Neither library requires an alternative signature, so both accept the chains downgraded by a quantum attacker, QAT1 and QAT2.

	Tested certificate chain	Bouncy Castle	wolfSSL
S1	all signatures valid	accept	accept
S2	end-entity certificate: valid native, invalid alternative signature	accept	reject
S3	end-entity certificate: invalid native, valid alternative signature	reject	reject
S4	valid, MCA extensions marked critical	reject	accept
S5	end-entity certificate: alternative key but no alternative signature, native valid; root and sub-CA full MCA	accept	accept
S6	end-entity certificate: valid alternative signature but no alternative key; root and sub-CA full MCA	accept	accept
S7	native-only root; sub-CA with alternative key, certified by the root’s native signature only; end-entity certificate with valid alternative signature by the sub-CA	accept	accept
S8	as S6, but end-entity alternative signature invalid	accept	reject
S9	as S7, but end-entity alternative signature invalid	accept	reject
S10	end-entity certificate: native signature with RIPEMD160withRSA, an algorithm wolfSSL lacks; alternative signature valid	accept	reject
S11	as S10, but no MCA extensions on the end-entity certificate (control)	accept	reject
S12	sub-CA: native signature with RIPEMD160withRSA by the root, alternative signature valid; end-entity certificate as in S1	accept	reject
S13	sub-CA: invalid native signature, valid alternative signature; end-entity certificate as in S1	reject	reject
QAT1	Quantum Attack 1: forged native-only sub-CA (signed with the root’s native key) under an alternative-capable root; native-only end-entity below it	accept	accept
QAT2	Quantum Attack 2: forged native-only end-entity (signed with the sub-CA’s native key) under a genuine alternative sub-CA	accept	accept

to Bouncy Castle it verifies the alternative signature during ordinary chain validation: its `CertManager` rebuilds the to-be-signed part and checks the alternative signature against the issuer's `subjectAltPublicKeyInfo`. The part it rebuilds (`wc_GeneratePreTBS`) omits the algorithm identifier of the `TBSCertificate`, so `wolfSSL`, too, implements the `PreTBSCertificate` of the Catalyst draft. This is why the two libraries interoperate in our tests, and why both deviate from the input as clarified by Corrigendum 3.

6.3 Test Results

As stated above, the test results that are presented in the following refer to the certificate path validation of both implementations. The tests confirm that `wolfSSL` rejects the same chain that Bouncy Castle accepts, namely one with a valid native and an invalid alternative signature (S2 in Table 2). It also rejects a chain whose alternative signature is valid but whose native signature is invalid (S3), so it verifies the native signature in every case and in effect requires both signatures to be valid even if an alternative signature is present. This is stricter than the ITU-T specification, which for a migrated relying party mandates only the validation of the alternative signature.

Chains S5 to S9 probe the strict reading of clause 7.2.2 (Section 1.3), under which a certificate with an alternative key must also carry an alternative signature and vice versa. `wolfSSL` enforces nothing of the kind. It accepts an end-entity certificate that carries an alternative key but no alternative signature (S5), one that carries a valid alternative signature but no alternative key (S6), and a chain whose native-only root has certified the sub-CA's alternative key by a native signature alone (S7). The two negative controls show that the alternative signature is verified in the latter two shapes rather than skipped: with an invalid alternative signature the same chains are rejected (S8, S9). `wolfSSL` thus implements the logical reading adopted in Section 1.3, and in S7 it trusts a sub-CA alternative key that no alternative signature has ever covered – the situation that a CA without an alternative key creates when it certifies a migrated subject. Bouncy Castle's path validation, which does not process the extensions, accepts all five chains.

Chains S10 to S13 ask the converse of the second point of Section 5: when the native signature cannot be verified, does a verifier fall back to the alternative signature? We give the end-entity certificate (S10) or the sub-CA certificate (S12) a native signature with `RIPEMD160withRSA`, an RSA signature algorithm that Bouncy Castle produces and verifies but that `wolfSSL`'s certificate code does not know, together with a valid alternative signature. `wolfSSL` rejects both with its error for an unknown signature algorithm, exactly as it rejects the native-only control (S11), and it likewise rejects a sub-CA whose native signature is invalid while its alternative signature is valid (S13), the sub-CA analogue of S3. `wolfSSL` thus never falls back to the alternative signature: the native signature is verified at every level and must verify. This means that a relying party running `wolfSSL` cannot accept an MCA certificate whose native algorithm it no longer supports, although the specification lets a migrated relying party verify the alternative signature alone. Thus traditional algorithms cannot, in this scenario, simply be withdrawn in `wolfSSL` per a build-configuration-based policy enforcement, which would be a reasonable safety measure once the quantum computer threat becomes real. If they were, certificates with traditional native algorithms and PQC signatures via the MCA mechanism would become unverifiable for it. Bouncy Castle, which supports `RIPEMD160withRSA`, accepts S10 to S12 and rejects S13 on the native signature.

The other widely used certificate libraries do not verify the MCA extensions. OpenSSL with the `oqs-provider` does not implement the alternative signature [RSP25]; Kim et al. [KCKK26] report that OpenSSL and the `oqs-provider` identify the extensions without enforcing them, and that GnuTLS, NSS, Go and the Python cryptography library accept on the native signature alone. Since the extensions are recommended to be non-critical, such a verifier silently ignores them and checks the native signature only, so the PQC

layer of an MCA hierarchy is invisible to it. Both recent studies [LSE⁺26, KCKK26] cover more implementations than we do and agree with our findings on Bouncy Castle and wolfSSL. Lee et al.’s control with a broken classical signature corresponds to our S3, and the native-only end-entity certificate that both studies find accepted corresponds in shape to our QAT2. Additional to both studies are in our test suite the criticality result (S4), the forged sub-CA of QAT1 and the sub-CA-level chains in general, since their certificate under test is always an end-entity certificate, the chains that probe the strict reading of clause 7.2.2 (S5 to S9), and the chains with a native signature the verifier cannot process (S10 to S13).

Finally, we examined the two attacks of Section 2 against both libraries. Against Bouncy Castle there is, strictly speaking, no downgrade: since it never verifies the alternative signature during path validation, it is effectively a traditional-only verifier, and accepting a native-only chain is its normal behaviour rather than a bypass. wolfSSL, on the other hand, does verify the alternative signature when one is present, yet it still accepts both forged chains – a native-only sub-CA (Attack 1) and a native-only end-entity certificate (Attack 2) under an alternative-capable issuer – exactly as the traditional-only Bouncy Castle does. The comparison is the point: enabling MCA support brings no security advantage against the quantum attacker, because the alternative signatures are verified only when present and never required, so the attacker simply omits them and presents a native-only chain.

7 Conclusion

The MCA extensions of ITU-T X.509 let a single certificate carry the native signature and an alternative signature at once, where in the intended use the former is a widely accepted legacy algorithm and the latter a modern algorithm the ecosystem is migrating to. The goal in the typical current use is that one certificate hierarchy can serve both non-migrated and migrated relying parties during the transition to PQC. The problem, however, is that the opportunistic signature verification mechanism implied by the MCA framework does not imply any gain in security over a mere traditional-only hierarchy. Specifically, the central and previously known problem is the lack of downgrade protection against a quantum attacker. A certificate chain of MCA certificates gives no advantage over a chain with purely traditional algorithms – their trust anchors equally must be removed from relying parties’ trust stores once the quantum threat becomes real. Only additional opt-in and pinning mechanisms could solve the problem. Together with the fact that per se the MCA mechanism does not provide any means to collect metrics about PQC adoption, its vulnerability to forgeries in the post-quantum era lets the mechanism add no value to an ecosystem migrating to PQC. The problem persists unchanged after all corrigenda and the amendment to the 2019 edition.

Our further findings about the formal EUF-CMA violations that are inherent to the MCA mechanism cannot be rated a practically relevant threat. As a more serious finding we rate the malleability problem arising when applying the ITU-T specification’s policy to verify only the alternative signature when possible, which allows arbitrary data modifications and bloat in the affected certificate fields of the validated certificate chain.

We also point out several points of verifier behaviour that the ITU-T specification of the MCA mechanism leaves unspecified and that may lead to divergent behaviour of applications in the field.

References

- [Ben26] Benjamin, David and DeBlasio, Joe. Post-Quantum HTTPS Authentication

- Roadmap, February 2026. Last updated 2026-02-27, <https://www.chromium.org/Home/chromium-security/post-quantum-auth-roadmap/>.
- [Bou] BouncyCastle JCE Provider. <http://www.bouncycastle.org/java.html>.
- [Bra97] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. BCP 14, RFC 2119, March 1997. <https://www.rfc-editor.org/rfc/rfc2119>.
- [Chr26] Chrome Secure Web Networking Team. Cultivating a Robust and Efficient Quantum-Safe HTTPS. Google Security Blog, February 2026. 2026-02-27, <https://blog.google/security/cultivating-a-robust-and-efficient-quantum-safe-https/>.
- [Coo] Cooper et al. RFC 5280. <http://tools.ietf.org/html/rfc5280>.
- [GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
- [HJB12] J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security (HSTS). RFC 6797, November 2012. <https://www.rfc-editor.org/rfc/rfc6797>.
- [Int19] International Telecommunication Union. Recommendation ITU-T X.509 (10/2019): Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks, 2019. <https://www.itu.int/rec/T-REC-X.509-201910-I>.
- [Int23] International Telecommunication Union. Recommendation ITU-T X.509 (2019) Corrigendum 1 (10/2021) and Corrigendum 2 (10/2023), 2023.
- [Int24] International Telecommunication Union. Recommendation ITU-T X.509 (2019) Amendment 1 (10/2024): Miscellaneous enhancements, 2024.
- [Int26] International Telecommunication Union. Recommendation ITU-T X.509 (2019) Corrigendum 3 (07/2026), prepublished version, 2026. Approved 2026-07-14; corrections of defect reports 445, 446, 449 and 450.
- [ISO21] ISO/IEC. ISO/IEC Directives, Part 2: Principles and rules for the structure and drafting of ISO and IEC documents. Ninth edition, 2021. <https://www.iso.org/directives-and-policies.html>.
- [KCKK26] Taesung Kim, Boheung Chung, Keonwoo Kim, and Yousung Kang. Classical Acceptance Is Not Hybrid Authentication: How Deployed X.509 Validation Stacks Treat Hybrid Certificates. arXiv:2607.20800v2, to appear in Computer Standards & Interfaces, 2026. <https://arxiv.org/abs/2607.20800>.
- [LSE⁺26] Minwoo Lee, Minjoo Sim, Siwoo Eum, Subeen Cho, Yulim Hyung, and Hwajeong Seo. Beyond Size: Do Hybrid PQC Certificates Actually Enforce the Classical-PQC Binding? A Cost-and-Security Study. *Cryptology ePrint Archive*, Paper 2026/1416, 2026. <https://eprint.iacr.org/2026/1416>.
- [Nat24] National Institute of Standards and Technology. Module-Lattice-Based Digital Signature Standard. FIPS PUB 204, August 2024. <https://doi.org/10.6028/NIST.FIPS.204>.
- [RSP25] Nino Ricchizzi, Christian Schwinne, and Jan Pelzl. Applied Post-Quantum Cryptography: A Practical Approach for Generating Certificates in Industrial Environments. arXiv:2505.04333, 2025. <https://arxiv.org/abs/2505.04333>.

- [SR26] Yaron Sheffer and Tirumaleswar Reddy.K. PQC Continuity: Downgrade Protection for TLS Servers Migrating to PQC (draft-sheffer-tls-pqc-continuity-02), 2026. Internet-Draft, work in progress, <https://datatracker.ietf.org/doc/draft-sheffer-tls-pqc-continuity/>.
- [TVF⁺18] A. Truskovsky, D. Van Geest, S. Fluhrer, P. Kampanakis, M. Ounsworth, and S. Mister. Multiple Public-Key Algorithm X.509 Certificates (draft-truskovsky-lamps-pq-hybrid-x509-01), 2018. Expired Internet-Draft, <https://datatracker.ietf.org/doc/draft-truskovsky-lamps-pq-hybrid-x509/>.
- [TVF⁺23] A. Truskovsky, D. Van Geest, S. Fluhrer, P. Kampanakis, M. Ounsworth, and S. Mister. Multiple Public-Key Algorithm X.509 Certificates (draft-truskovsky-lamps-pq-hybrid-x509-02), 2023. Expired Internet-Draft, final version of 2023-08-24, <https://datatracker.ietf.org/doc/draft-truskovsky-lamps-pq-hybrid-x509/02/>.
- [wol24] wolfSSL Inc. X.509 Alternative Public Key and Signature (Dual Algorithm / Chimera Certificates), 2024. <https://www.wolfssl.com/x-509-alternative-public-key-and-signature/>.

A Changes of Corrigendum 3 to the MCA-Related Text of ITU-T X.509 (2019)

This appendix reproduces the passages of ITU-T X.509 (10/2019) [Int19] that concern the MCA mechanism and that Corrigendum 3 (07/2026) [Int26] changes, as a marked-up diff: text that the corrigendum deletes is shown ~~red and struck out~~, text that it adds is shown green and underlined. Text of ITU-T X.509 is set in boxes, each headed by the clause it is taken from. The base text is the 2019 edition. Corrigenda 1 and 2 and Amendment 1 do not touch these passages, except that Amendment 1 rewrote the import section of the Annex A module that Section A.5 extends. Items 6, 7 and 9 of the corrigendum, which resolve defect reports 445, 446 and 450, are covered. Item 8 (defect report 449) restructures the ALGORITHM information object class of clause 6.2.2 and is not specific to the MCA mechanism. The corrigendum refers to clauses 9.9.2 and 9.9.3 where the 2019 edition numbers them 9.8.2 and 9.8.3, and for clause 9.9.3 it names the `subjectAltPublicKeyInfo` extension where it means `altSignatureAlgorithm`. The 2019 numbering and the evident meaning are used below. Prose and ASN.1 are reproduced verbatim from the 2019 edition, including its typography.

A.1 Clause 6.2.1, the SIGNED Data Type (Defect Report 446)

ITU-T X.509 (10/2019), clause 6.2.1, as changed by Corrigendum 3

If ~~one or two signatures~~ a signature ~~are~~ is to be appended to the data, the following ASN.1 may be used to define the data type resulting from applying ~~one or two signatures~~ a signature to the given data type.

~~NOTE—The components after the extension mark were added from Rec. ITU-T X.509 (2016) + ISO/IEC 9594-8:2017.~~

```
SIGNED{ToBeSigned} ::= SEQUENCE {
    toBeSigned ToBeSigned,
    COMPONENTS OF SIGNATURE,
    ...7 }
++4+
```

```
altAlgorithmIdentifier AlgorithmIdentifier{{SupportedAltAlgorithms}} OPTIONAL,  
altSignature BIT STRING OPTIONAL]]  
}(WITH COMPONENTS {..., altAlgorithmIdentifier PRESENT, altSignature PRESENT } |  
  WITH COMPONENTS {..., altAlgorithmIdentifier ABSENT, altSignature ABSENT } ) }
```

A.2 Clauses 7.2.1, 7.10.2, 11.3 and 14.2 (Defect Report 446)

The first paragraph after the `ExtensionSet` definition in clause 7.2.1, after the `crlExtensions` definition in clause 7.10.2 and after the `IssuerSerialNumber` definition in clauses 11.3 and 14.2 reads identically in all four places and is deleted:

ITU-T X.509 (10/2019), clauses 7.2.1, 7.10.2, 11.3 and 14.2, as changed by Corrigendum 3
~~When generating a digital signature using the SIGNED parameterized data type,
 only one digital signature shall be generated, i.e., the altAlgorithmIdentifier and the
 altSignature components shall be absent.~~

A.3 Clause 7.2.2 (Defect Report 450)

ITU-T X.509 (10/2019), clause 7.2.2, as changed by Corrigendum 3

It is possible for a public-key certificate to hold:

- a subject's alternative public key information to be used instead of the information provided in the `subjectPublicKeyInfo` component;
- an alternative signature algorithm to be used instead of the algorithm specified in the `signature` component of the TBSCertificate data type; and
- an alternative digital signature to be checked instead of the native digital signature.

Such alternative specifications are provided as public-key certificate extensions. [...]

A CA generating a public-key certificate with the alternative cryptographic algorithms and alternative digital signature shall:

- when generating the value in the `altSignatureValue` extension, exclude the `signature` component of the SIGNED data type and the `altSignatureValue` extension from the public-key certificate, and generate the digital signature over the remaining DER encoded public-key certificate using the algorithm specified in the `altSignatureAlgorithm` extension; and
- when generating the value in the `signature` component of the SIGNED data type, the `subjectAltPublicKeyInfo`, the `altSignatureAlgorithm` and the `altSignatureValue` extensions shall be included in the DER encoding of the public-key certificate.

A.4 Clauses 9.8.2 and 9.8.3 (Defect Report 445)

In clause 9.8.2, the syntax of the subject alternative public key information extension changes to:

ITU-T X.509 (10/2019), clause 9.8.2, as changed by Corrigendum 3

```
SubjectAltPublicKeyInfo ::= SEQUENCE {
    algorithm AlgorithmIdentifier{{SupportedAlgorithmsSupportedAltPublicKeyAlgorithms}},
    subjectAltPublicKey BIT STRING }
```

In clause 9.8.3, the syntax of the alternative signature algorithm extension and its NOTE 2 change to:

ITU-T X.509 (10/2019), clause 9.8.3, as changed by Corrigendum 3

```
AltSignatureAlgorithm ::=
    AlgorithmIdentifier{{SupportedAlgorithmsSupportedAltSignatureAlgorithms}}

[...]
```

NOTE 2 – It is recommended that it be flagged as non-critical. Flagging it as critical would require all relying parties to understand the extension and the alternative ~~public-key~~ digital signature algorithms.

A.5 Annex A, Module CertificateExtensions (Defect Report 445)

The two ASN.1 changes of Section A.4 apply equally to the module. Its import section, as rewritten by Amendment 1, receives the additional statement

ITU-T X.509 (10/2019), Annex A, module CertificateExtensions, import section, as changed by Corrigendum 3

```
SupportedAltPublicKeyAlgorithms, SupportedAltSignatureAlgorithms,
SupportedAttributes
FROM SupportedInformationObjects
{joint-iso-itu-t ds(5) module(1) supportedInformationObjects(49) 10} WITH
SUCCESSORS
```

and the statement

ITU-T X.509 (10/2019), Annex A, module CertificateExtensions, as changed by Corrigendum 3

```
SupportedAttributes ATTRIBUTE ::= {...}
```

is commented out. The latter statement is not part of the 2019 edition, whose module imports SupportedAttributes from InformationFramework; Amendment 1 replaced that import.

B Control Flow of Algorithm 1

Figure 6 shows the control flow of Algorithm 1 (Section 2.7) as a flowchart. The two highlighted decisions are the only ones that depend on the policy of Section 2.6; every other rejection is unconditional, since it concerns a field of the subject certificate that the attacker controls.

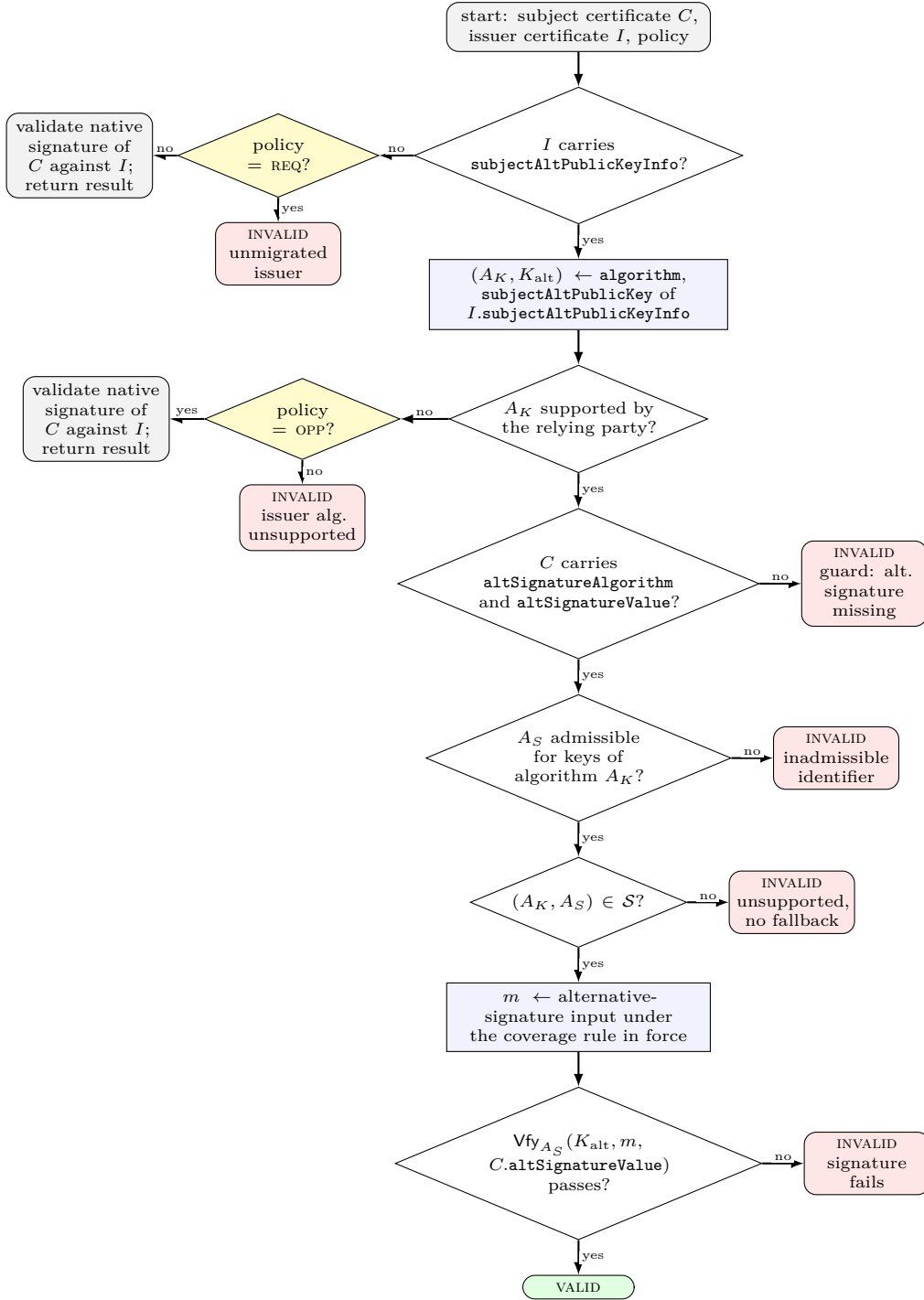


Figure 6: Control flow of Algorithm 1. Diamonds are decisions, the two highlighted ones depend on the policy (REQ = Require-PQC, OPP = Opportunistic-PQC); rounded boxes are terminals. “Validate native signature” returns the result of the native validation of C against $I.subjectPublicKeyInfo$, as in lines 1 and 9 of the algorithm.