Public Key Linting for ML-KEM and ML-DSA

Evangelos Karatsiolis[1] Franziskus Kiefer[2] Juliane Krämer[3] Mirjam Loiero[1]
Christian Tobias[1] Maximiliane Weishäupl[3]

SPIQE, 24 June 2025

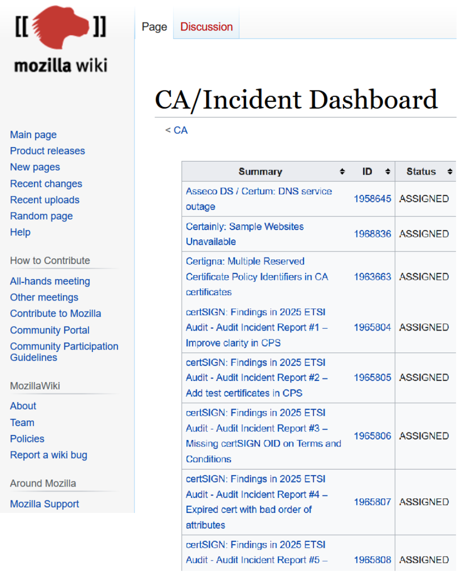
[1]MTG

[2]Cryspen

[3]Universität Regensburg

Motivation

Incident Dashboard



[[ ]]
mozilla wiki

Main page
Product releases
New pages
Recent changes
Recent uploads
Random page
Help

How to Contribute

All-hands meeting
Other meetings
Contribute to Mozilla
Community Portal
Community Participation Guidelines

MozillaWiki

About
Team
Policies
Report a wiki bug

Around Mozilla

Mozilla Support

Page Discussion

CA/Incident Dashboard

< CA

Summary	ID	Status
Asseco DS / Certum: DNS service outage	1958645	ASSIGNED
Certainly: Sample Websites Unavailable	1968836	ASSIGNED
Certigna: Multiple Reserved Certificate Policy Identifiers in CA certificates	1963663	ASSIGNED
certSIGN: Findings in 2025 ETSI Audit - Audit Incident Report #1 – Improve clarity in CPS	1965804	ASSIGNED
certSIGN: Findings in 2025 ETSI Audit - Audit Incident Report #2 – Add test certificates in CPS	1965805	ASSIGNED
certSIGN: Findings in 2025 ETSI Audit - Audit Incident Report #3 – Missing certSIGN OID on Terms and Conditions	1965806	ASSIGNED
certSIGN: Findings in 2025 ETSI Audit - Audit Incident Report #4 – Expired cert with bad order of attributes	1965807	ASSIGNED
certSIGN: Findings in 2025 ETSI Audit - Audit Incident Report #5 –	1965808	ASSIGNED

- **c**ertification **a**uthorities (**CA**) issue certificates used to verify the identity of entities
  - crucial for security and functioning of protocols like TLS, S/MIME, . . .
  - essential to establish trust among users
- several requirements regarding certification exist
  → failing to comply with them leads to **incidents**

Incident Dashboard

GlobalSign: Non-BR-Compliant Certificate Issuance -- RSA key smaller than 2048 bits	1393557	RESOLVED
Sectigo: Certificates with RSA keys where modulus is not divisible by 8	1653504	RESOLVED
GDCA: Misissuance of certificates with small RSA keys	1467414	RESOLVED

- incidents harm a CA's reputation and more generally public key infrastructures
- many incidents are related to the **content of a certificate**
- can be avoided, if proper mechanisms are in place

Linting

- **linting:** process of analyzing the content of a certificate w.r.t. predefined rules ("**lints**")
- examples for RSA lints:
  - the length of the modulus is one of the specified values in FIPS 186-3
  - the modulus and the public exponent are odd numbers
  - the modulus is not a power of a prime
  - the modulus has no factors smaller than 752

Lint Example

Lint: The modulus has no factors smaller than 752

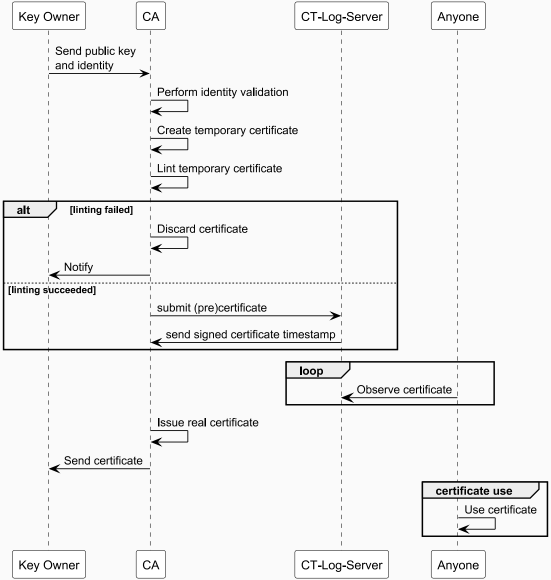
Check that $\gcd(n, r) = 1$ where

$$\begin{aligned} r =& 1451887755777639901511587432083070202422614380984889313550570919 \\ & 6593151770659565743590789126541491676439926842369913057775743308 \\ & 3166651158914570105971074227669275788291575622090199821297575654 \\ & 3223550490431013061082131040808010565293748926901442915057819663 \\ & 730454818359472391642885328171302299245556663073719855\,, \end{aligned}$$

is the product of the 132 primes from 3 to 751.

Linting Process

- CA creates temporary certificate and starts the linting process
- certificate is checked for each lint in the linting library
- **if one lint fails**, the entire linting process fails → the CA discards the certificate and notifies the key owner
- **otherwise:** certificate is sent to a certificate transparency log server, finalized, and issued



Current State of Linting Regulations

Ballot SC-75 of CA/Browser Forum...

- ...renders pre-sign linting **mandatory** from March 2025
- ...states that validating the key material is a **responsibility** of the CA

Currently: these requirements cover RSA and elliptic curve keys

Linting Post-Quantum Schemes

- advancing standardization of **p**ost-**q**uantum **c**ryptographic (**PQC**) schemes:
  - finalized standards: ML-KEM, ML-DSA, SLH-DSA
  - also selected for standardization: Falcon, HQC
  - ongoing NIST process for additional digital signature schemes
- need for preparing the IT security infrastructure for integrating post-quantum schemes
- linting has not been studied for PQC schemes!

Goal: initiate the study of linting for PQC schemes by analyzing the public keys of ML-KEM and ML-DSA

Background: ML-KEM

ML-KEM: General Information

- NIST standard for **k**ey-**e**ncapsulation **m**echanisms (**KEM**) using module lattices
- based on the KEM CRYSTALS-Kyber
- high-level construction:

$$\text{public-key encryption scheme: K-PKE} \xrightarrow{\text{FO-transform}} \text{KEM: ML-KEM}$$

- distinction between external and internal components:
  - external: generate randomness, check whether randomness generation was successful, and call their internal counterparts
  - internal: actual steps of the procedures ← here the public key is used!

ML-KEM: Key Generation

Algorithm ML-KEM.KeyGen_internal(d, z)

Input randomness $d, z \in \mathbb{B}^{32}$

Output $ek \in \mathbb{B}^{384k+32}$, $dk \in \mathbb{B}^{768k+96}$

1: $(ek_{\text{PKE}}, dk_{\text{PKE}}) \leftarrow$ K-PKE.KeyGen(d)

2: $ek \leftarrow ek_{\text{PKE}}$

3: $dk \leftarrow (dk_{\text{PKE}} \| ek \| H(ek) \| z)$

4: **return** (ek, dk)

ML-KEM: Key Generation of the underlying PKE

Algorithm K-PKE.KeyGen(d)

Input: randomness $d \in \mathbb{B}^{32}$

Output: $ek_{\text{PKE}} \in \mathbb{B}^{384k+32}$, $dk_{\text{PKE}} \in \mathbb{B}^{384k}$

$(\rho, \sigma) \leftarrow G(d||k)$

$N \leftarrow 0$

for $(i \leftarrow 0; i < k; i{+}{+})$ **do**

  for $(j \leftarrow 0; j < k; j{+}{+})$ **do**

    $\hat{A}[i, j] \leftarrow \text{SampleNTT}(\rho||j||i)$

  end for

end for

for $(i \leftarrow 0; i < k; i{+}{+})$ **do**

  $s[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(\sigma, N))$

  $N \leftarrow N + 1$

end for

Algorithm K-PKE.KeyGen(d)

for $(i \leftarrow 0; i < k; i{+}{+})$ **do**

  $e[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(\sigma, N))$

  $N \leftarrow N + 1$

end for

$\hat{s} \leftarrow \text{NTT}(s)$

$\hat{e} \leftarrow \text{NTT}(e)$

$\hat{t} \leftarrow \hat{A} \circ \hat{s} + \hat{e}$

$ek_{\text{PKE}} \leftarrow \text{ByteEncode}_{12}(\hat{t})||\rho$

$dk_{\text{PKE}} \leftarrow \text{ByteEncode}_{12}(\hat{s})$

return $(ek_{\text{PKE}}, dk_{\text{PKE}})$

Methodology

Finding Lints

- consider properties of the certificate and the public key itself
- check whether all rules from the standards are fulfilled and if the properties of an honestly generated key are given
- take into account the input validation checks described in the standards

Lint Classes

We introduce 5 lint classes:

- **INTER:** interoperability lints focus on certificate properties which assist applications in properly communicating with each other
- **DIM:** dimension lints test whether the size of certain objects is correct
- **DOM:** domain lints test whether the “type” of the objects is correct
- **DIS:** distribution lints verify distribution properties of objects
- **GEN:** lints that work generically for any scheme

Completeness of Lints

- lint classes: new ones might need to be introduced for future lints!
- dimension and domain lints: complete
- distribution lints: incomplete
- interoperability and generic lints: incomplete

→ only starting point for post-quantum linting
→ more lints can be added over time using our proposed formal description

Application Lints

- applications (e.g., email client, browser) extract the pk from the certificate in order to use it
- in the case of ML-KEM and ML-DSA, the application must perform further operations, like expanding the key
  $\rightarrow$ the expanded pk depends on the implementation of the expanding algorithm used by the application
  $\rightarrow$ lints for the expanded pk must be performed at the application side (if at all)

Implementation

- our lints are implemented in Java and Rust
  - lints which examine properties of the certificate $\rightarrow$ Java (using methods provided by BouncyCastle)
  - lints that check the correctness of the key within the certificate $\rightarrow$ Rust
- Rust allows us to enforce the size of inputs on the type level, such that dimension lints get covered by the API
- we give test vectors for keys with and without errors

ML-KEM Lints

Overview

Lint	Classification	Identifier
key usage	interoperability	INTER_01
pk aid enc	interoperability	INTER_02
ek length	dimension	DIM_01
ek seed length	dimension	DIM_02
ek matrix dimension	dimension	DIM_03
ek vector dimension	dimension	DIM_04
ek matrix entries	domain	DOM_01
ek vector entries	domain	DOM_02
ek seed entry frequency	distribution	DIS_01
ek seed entry run	distribution	DIS_02
ek seed small/large entries	distribution	DIS_03
ek matrix entry frequency	distribution	DIS_04
ek matrix entry run	distribution	DIS_05
ek matrix small/large entries	distribution	DIS_06
known enc key	generic	GEN_01
ek encoding	generic	GEN_02

Example: Interoperability Lint

ML_KEM_INTER_01

- concerns the key usage extension in a certificate, which specifies for which types of use the certificate's public key can be used
- lint checks that the value of the key usage extension is compatible with the ML-KEM algorithm
- only values related to key encryption are compatible

Example: Dimension Lint

ML_KEM_DIM_01

- based on one of the tests for input validation described in the ML-KEM standard
- lint checks the length of the encoded encapsulation key *ek*
- the correct length is $384k + 32$ bytes for $k \in \{2, 3, 4\}$ depending on the ML-KEM parameter set

Example: Distribution Lint

ML_KEM_DIS_01

- checks if the seed ρ contains the same element an amount of times that is unlikely for a pseudorandomly sampled value from $\mathbb{B}^{32}$
- outputs an error if there are at least $x = 20$ occurrences of the same byte

Conclusion

Conclusion

- we initiate the study of PQC linting and provide a framework
- challenges:
  - for PQC schemes, security is often related to certain elements "looking random" or following a pre-defined **distribution**
  - usage of **seeds** in ML-KEM and ML-DSA: properties of the expanded public key cannot be checked by the CA (depends on the implementation of the expansion algorithm)
    $\rightarrow$ testing of the implementations is necessary
- only a starting point: linting is very scheme-specific!