Post-Quantum Cryptography Algorithm Cheat Sheets

Symbols

General color coding

Best to Worst: ■ ■ ■ ■ ■ TBD: ■

Symbols

- Recommendations differ depending on organization
- Parameter set
- Key Generation
- Encryption
- Decryption
- Signing
- Verification
- Not yet standardized by NIST
- CPU Cycles
- Implementation Code
- Implementation complexity
- Implementation size

Security categories of parameter sets

V IV III II I — NIST Security Categories V, IV, III, II, I

Higher means more secure.

Implementation complexity and size

L M H — Low/Medium/High implementation complexity

L M H — Low/Medium/High implementation size

Lower is better.

Rating scales for parameter sizes and performance

Best to Worst: ■ $n \leq 2$, ■ $n\{3,4\}$, ■ $n \in \{5,6\}$, ■ $n \in \{7,8\}$, ■ $n \geq 9$

- n $\mathcal{O}(5^n)$ CPU kilo cycles for key generation
- n $\mathcal{O}(5^n)$ CPU kilo cycles for signing
- n $\mathcal{O}(5^n)$ CPU kilo cycles for signature verification
- n $\mathcal{O}(5^n)$ CPU kilo cycles for encryption / key encapsulation
- n $\mathcal{O}(5^n)$ CPU kilo cycles for decryption / key decapsulation
- n $\mathcal{O}(2^n)$ KB of signature size
- n $\mathcal{O}(2^n)$ KB of ciphertext size
- n $\mathcal{O}(2^{(n-5)})$ KB of signature algorithm public key size
- n $\mathcal{O}(2^n)$ KB of encryption algorithm public key size

How To Interpret This Cheat Sheet

The goal of this cheat sheet is to make it as easy as possible to figure out which algorithm to pick for a given use case. Algorithm ID cards break down algorithm parameter sets, their important values and performance characteristics. The cheat sheet is intended to help users primarily in technical roles, such as engineers, architects or software developers working with post-quantum cryptography.

The focus is to avoid giving specific numbers measured in bits, bytes or cycles as this makes makes comparing numbers across algorithms difficult. Instead, this complexity is simplified by only providing a **color-coded number indicating the order of magnitude of each metric**.

This approach prioritizes easy interpretation and comparability of metrics and in general quick informational gain over absolute precision of data – remember this is a cheat sheet, not a standard! This document is not intended to replace the study of algorithm specifications. It just aims to point you in the right direction quickly.

The approach of focusing on orders of magnitude walks a fine line between treating too many things as "equal" and not simplifying things enough to be easy to read and compare. "In the same order of magnitude" usually refers to "equal up to a factor of at most 10", which is a very coarse way of comparing numbers. Treating metrics that differ by a factor of e.g. 9.9 as "equal" because 9.9 < 10 paints a distorted picture. In cryptography, factors of 5 or even 2 can make a significant difference in performance, both in theory and in practice. In order to still tease out the differences in metrics without throwing too many things together that actually differ significantly, this cheat sheet applies different scaling and "orders of magnitude" (i.e., not regarding base 10) for different metrics.

It turns out that for **metrics measured in (kilo) CPU cycle counts, i.e. algorithm performance, "up to a factor of 5"** is a scale that is granular enough to work out the differences between algorithms while maintaining easy comparability. Those cycle counts heavily depend on the CPU used during measurement, hence the numbers need to be taken with a grain of salt, even if given exactly and not in terms of orders of magnitude.

For **signature and ciphertext sizes as well as key sizes, measuring numbers in kilobytes "up to a factor of 2"** is well suited to work out the differences between algorithms while allowing for quick comparison. Specifically for signature public key sizes only, we offset the corresponding color coding by 5 orders of magnitude. This is because SLH-DSA has extremely small pubic keys compared to all other signature algorithms, which would extend the scale into negative numbers (e.g. for SLH-DSA-SHA2-128s, the public key has $32=2^5$ bytes, which corresponds to an order of magnitude of -5 when measuring in orders magnitude of factor 2 and in kilobytes). This phenomenon of algorithm metrics spanning a very large range of orders of (base 2) magnitudes does not occur to this extent for encryption algorithms, making an offset unnecessary.

All values thus have a lower bound of 0. We do not limit the upper end of scales, but don't distinguish values greater than 10 anymore in terms of color coding. Please refer to the definitions on the left for symbol explanations, color coding and interpretation of numeric values.

Post-Quantum Cryptography Algorithm Cheat Sheets

Algorithm Choice Guidance: Rules Of Thumb

The following orders of preference should be considered as general rules of thumb applicable to most use cases. They should however not be considered to be universal truths. Unstandardized algorithms are listed for the sake of completeness.

Signature Algorithms

Objective: Best All-Around Package TBV

ML-DSA → Falcon → SLH-DSA → XMSS/LMS* → New Candidates**

Objective: Performance of Key Generation TBV

ML-DSA → SLH-DSA f variants → SLH-DSA s variants → Falcon

Objective: Performance of Signing TBV

ML-DSA → Falcon → SLH-DSA f variants → SLH-DSA s variants

Objective: Performance of Verification TBV

Falcon → ML-DSA → SLH-DSA s variants → SLH-DSA f variants

Objective: Small Signatures TBV

Falcon → ML-DSA → SLH-DSA s variants → SLH-DSA f variants

Objective: Small Public Keys TBV

SLH-DSA → Falcon → ML-DSA

* XMSS/LMS is not suitable for general purpose signatures
** Security still unclear, none recommended yet at the moment

Encryption/Key Encapsulation Algorithms

Objective: Best All-Around Package TBV

ML-KEM → HQC → C. McEliece → BIKE → Frodo-KEM

Objective: Performance of Key Generation TBV

ML-KEM → HQC → BIKE → Frodo-KEM → C. McEliece

Objective: Performance of Encryption TBV

C. McEliece → ML-KEM → BIKE → HQC → Frodo-KEM

Objective: Performance of Decryption TBV

ML-KEM → C. McEliece → HQC → Frodo-KEM → BIKE

Objective: Small Ciphertext TBV

C. McEliece → ML-KEM → BIKE → HQC → Frodo-KEM

Objective: Small Public Keys TBV

ML-KEM → BIKE → HQC → Frodo-KEM → C. McEliece

Pure PQC vs PQ/T Hybrid

This topic depends on too many factors (e.g. cost of migration, security considerations, risk profile, GRC requirements) to give general advice. **For PQ/T hybrids, consider ECC (e.g. secp256r1, Curve25519)** over RSA for the traditional component.

Security Category Choices

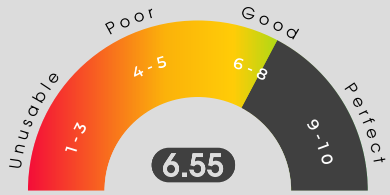
- First, consider using III as a baseline.
- Use IV or V for more security if possible (i.e., if a decrease in performance is not a concern and if no constraints apply).
- Use I or II **if and only if** III or higher is not an option due to constraints (e.g. performance, memory, etc.).

Pure vs. Pre-Hashing

- **First, consider using pure** (i.e., without pre-hashing) as this is the general recommendation.
- **Pre-Hashing may be considered if one or more of the following applies:**
  - The message M is too large to be sent to cryptographic module (CM) for hashing without significantly impacting performance. This may be the case e.g. in CMS related use cases such as S/MIME or code signing, or in cases of very narrow communication channels to the CM (e.g. between APDUs exchanged between smartcard and smartcard reader).
  - The hash needs to be signed with different algorithms and would be computed repeatedly without pre-hashing.
  - The specific hash function is not supported in a CM.

Signature Algorithm Overview & ID Cards

ML-DSA (Module-Lattice-Based Digital Signature Algorithm)



Algorithm Overall Usability Score

Parameter Set	OID	Security Category	Performance (Key Generation)	Performance (Signing)	Performance (Verification)	Signature Size	Public Key Size	Suitable Pre-Hashing
ML-DSA-44	2.16.840.1.101.3.4.3.17	II	3	3	2	1	5	SHA-256, SHA3-256
ML-DSA-65	2.16.840.1.101.3.4.3.18	III	3	3	3	1	5	SHA-384, SHA3-384
ML-DSA-87	2.16.840.1.101.3.4.3.19	V	4	4	3	2	6	SHA-512, SHA3-512

Previous Name	CRYSTALS-DILITHIUM
Specification	FIPS 204
Type	Signature
Family	Lattice
Standardization Status	Standardized
Recommended By	NIST, BSI, ANSSI
Hashing	Pure, Pre-Hashing
Naming	by $k \times l$ matrix A (e.g. $6 \times 5 \rightarrow$ ML-DSA-65)

Algorithm Implementation	Key Generation	Signing	Verification
Complexity	?	?	?
Size	?	?	?

/ Pros / Use If:

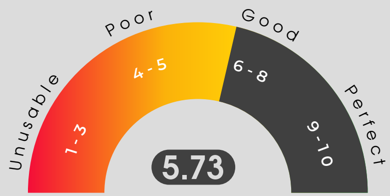
- You need a general-purpose signature algorithm with decent specs in all categories

/ Cons / Don't Use If:

- You don't want a lattice-based algorithm

Note: Cycle counts for key generation, signing and verification depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

Falcon (Fast-Fourier Lattice-Based Compact Signatures over NTRU)



Algorithm Overall Usability Score

Previous Name	Falcon
Specification	Project Page
Type	Signature
Family	Lattice
Standardization Status	Pending
Recommended By	TBD
Hashing	TBD
Naming	TBD

Algorithm Implementation	Key generation	Signing	Verification
Complexity	H	H	?
Size	?	?	?

Parameter Set	OID	Security Category	Performance (Key generation)	Performance (Signing)	Performance (Verification)	Signature Size	Public Key Size	Suitable Pre-Hashing
Falcon-512	TBD	I	8	4	2	0	4	TBD
Falcon-1024	TBD	V	9	4	3	0	5	TBD

/ Pros / Use If:

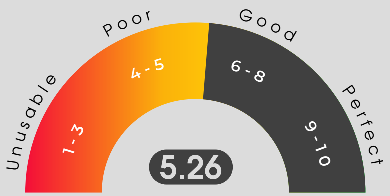
- Falcon has smaller signature sizes than ML-DSA: 0 vs. 1 resp. 2
- Falcon has smaller public key sizes than ML-DSA: 4 vs. 5 on Level I, 5 vs. 6 on Level V.

/ Cons / Don't Use If:

- You need a medium security category between I and V
- The algorithm is not yet standardized
- The algorithm requires expensive floating point arithmetic
- Key generation and signing are slower than for ML-DSA

Note: Cycle counts for key generation, signing and verification depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

SLH-DSA (Stateless Hash-Based Digital Signature Standard)



Algorithm Overall Usability Score

Previous Name	SPHINCS+
Specification	FIPS 205
Type	Signature
Family	Hash (stateless)
Standardization Status	Standardized
Recommended By	NIST, BSI, ANSSI
Hashing	Pure, Pre-Hashing
Naming	based on various characteristics (*s=small signatures, *f=fast)

Algorithm Implementation	🔑	✏	?
Complexity	?	?	?
Size	?	?	?

Parameter Set	OID	Security Category	Performance 🔑	Performance ✏	Performance ?	Signature Size	Public Key Size	Suitable Pre-Hashing
SLH-DSA-SHA2-128s	2.16.840.1.101.3.4.3.20	I	8	9	5	2	0	SHA-256, SHA3-256
SLH-DSA-SHA2-128f	2.16.840.1.101.3.4.3.21	I	5	7	5	4	0	SHA-256, SHA3-256
SLH-DSA-SHA2-192s	2.16.840.1.101.3.4.3.22	III	8	9	5	4	0	SHA-384, SHA3-384
SLH-DSA-SHA2-192f	2.16.840.1.101.3.4.3.23	III	5	7	6	5	1	SHA-384, SHA3-384
SLH-DSA-SHA2-256s	2.16.840.1.101.3.4.3.24	V	7	9	5	4	0	SHA-512, SHA3-512
SLH-DSA-SHA2-256f	2.16.840.1.101.3.4.3.25	V	6	8	6	5	1	SHA-512, SHA3-512
SLH-DSA-SHAKE-128s	2.16.840.1.101.3.4.3.26	I	8	9	5	2	0	SHA-256, SHA3-256
SLH-DSA-SHAKE-128f	2.16.840.1.101.3.4.3.27	I	5	7	5	4	0	SHA-256, SHA3-256
SLH-DSA-SHAKE-192s	2.16.840.1.101.3.4.3.28	III	8	9	5	4	0	SHA-384, SHA3-384
SLH-DSA-SHAKE-192f	2.16.840.1.101.3.4.3.29	III	5	7	6	5	1	SHA-384, SHA3-384
SLH-DSA-SHAKE-256s	2.16.840.1.101.3.4.3.30	V	7	9	5	4	0	SHA-512, SHA3-512
SLH-DSA-SHAKE-256f	2.16.840.1.101.3.4.3.31	V	6	8	6	5	1	SHA-512, SHA3-512

/ Pros / Use If:

- Alternative to ML-DSA and Falcon that is not based on lattices
- Very small public keys

/ Cons / Don't Use If:

- Poor key generation and signing performance compared to other algorithms
- High complexity of the algorithm and the implementation
- Possible interoperability issues due to the many variants that may not all be supported everywhere

Note: Cycle counts for key generation, signing and verification depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

XMSS / XMSS-MT (eXtended Merkle Signature Scheme / eXtended Merkle Signature Scheme Multi Tree)

Algorithm Overall Usability Score

Previous Name: XMSS/XMSSMT
Specification: SP 800-208, RFC 8391
Type: Signature
Family: Merkle Trees (stateful hash trees)
Standardization Status: Standardized
Recommended By: NIST, BSI, ANSSI
Hashing: TBD
Naming: XMSS-[Hashfamily]_[h]_[n]
XMSSMT-[Hashfamily]_[h]/[d]_[n]
where h is the tree height,
d is the number of layers, and
n is the message length in bits

Algorithm Implementation			
Complexity	?	?	?
Size	?	?	?

Parameter Set	Numeric Identifier	Security Category	Performance			Signature Size	Maximum Signatures	Number of Layers
XMSS-SHA2_10_256	0x00000001	V	?	?	?	?	2^{10}	1
XMSS-SHA2_16_256	0x00000002	V	?	?	?	?	2^{16}	1
XMSS-SHA2_20_256	0x00000003	V	?	?	?	?	2^{20}	1
XMSSMT-SHA2_20/2_256	0x00000001	V	?	?	?	?	2^{20}	2
XMSSMT-SHA2_20/4_256	0x00000002	V	?	?	?	?	2^{20}	4
XMSSMT-SHA2_40/2_256	0x00000003	V	?	?	?	?	2^{40}	2
XMSSMT-SHA2_40/4_256	0x00000004	V	?	?	?	?	2^{40}	4
XMSSMT-SHA2_40/8_256	0x00000005	V	?	?	?	?	2^{40}	8
XMSSMT-SHA2_60/3_256	0x00000006	V	?	?	?	?	2^{60}	3
XMSSMT-SHA2_60/6_256	0x00000007	V	?	?	?	?	2^{60}	6
XMSSMT-SHA2_60/12_256	0x00000008	V	?	?	?	?	2^{60}	12

Note: SP 800-208 defines further parameter sets not listed in RFC 8391 using other hash functions (SHA256/192, SHAKE256/256, SHAKE256/192). Furthermore, RFC 8391 lists optional parameter sets that are not approved in SP 800-208. All of those variants are omitted here as they are not likely to be widely used, in particular not after ML-DSA and SLH-DSA have been standardized in the meantime.

/ Pros / Use If:

- You can predict the maximum number of signatures that are going to be required
- Firmware signing use cases
- You want a signature scheme where the security only relies on the security of the hash function used without assuming the hardness of another mathematical problem.
- Cf. SP 800-208, Section 1.1 for additional explanations

/ Cons / Don't Use If:

- You require an algorithm for general use
- You cannot predict the maximum number of signatures that are going to be required, or the number of required signatures exceeds the maximum number of signatures enabled through the approved parameter sets
- Your application does not allow for the careful state management and tracking of signatures performed that is required with this algorithm

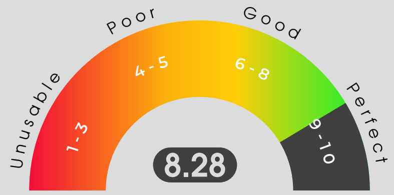
Note: Cycle counts for key generation, signing and verification depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

LMS (Leighton-Micali Signature)

TBD

Encryption Algorithm Overview & ID Cards

ML-KEM (Module-Lattice-Based Key-Encapsulation Mechanism Standard)



Algorithm Overall Usability Score

Parameter Set	OID	Security Category	Performance (key generation)	Performance (encryption)	Performance (decryption)	Ciphertext Size	Public Key Size
ML-KEM-512	2.16.840.1.101.3.4.4.7	I	2	2	2	0	0
ML-KEM-768	2.16.840.1.101.3.4.4.2	III	2	2	3	0	0
ML-KEM-1024	2.16.840.1.101.3.4.4.3	V	3	3	3	0	0

Previous Name CRYSTALS-KYBER
Specification FIPS 203
Type Encryption/KEM
Family Lattice
Standardization Status Standardized
Recommended By NIST, BSI, ANSSI
Naming TBD

Algorithm Implementation	(key)	(edit)	(?)
Complexity	?	?	?
Size	?	?	?

/ Pros / Use If:

- Currently the only post-quantum encryption /key encapsulation algorithm standardized by NIST
- Need a general-purpose encryption / key-encapsulation algorithm with decent specs in all categories

/ Cons / Don't Use If:

- You don't want a lattice-based algorithm

Note: Cycle counts for key generation, encryption and decryption depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

Classic McEliece

TBD

Note: Cycle counts for key generation, encryption and decryption depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

BIKE

TBD

Note: No data available for BIKE-L5 for cycle counts. Algorithm score is computed over BIKE-l1 and BIKE-L3 only.

Note: Cycle counts for key generation, encryption and decryption depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

HQC

TBD

Note: Cycle counts for key generation, encryption and decryption depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

FrodoKEM

TBD

Note: Cycle counts for key generation, encryption and decryption depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

Algorithm Scoring: Algorithm Overall Usability Score

We try to measure an algorithm's overall usability for a general use case without any special characteristics by calculating a single number between 0 (worst) and 10 (best)for the algorithm. This calculation is taking into account all performance metrics, public key size, signature/ciphertext size, the number of security categories provided, whether or not it is suitable for general use, and the complexity and size of its implementation.
We define an algorithm's overall score as

$$\text{score}_{\textit{algorithm}} = \max\left\{0;\ \text{avg}\{\text{score}_v \mid v \text{ is variant of } \textit{algorithm}\} - \frac{1}{8}\cdot\left(5 - \text{categories}_{\textit{algorithm}}\right) - \text{general}_{\textit{algorithm}} - \frac{1}{8}\cdot\text{impl}_{\textit{algorithm}}\right\}$$

where $\text{score}_{\textit{variant}}$ is a score for an individual algorithm variant (parameter set) computed as

$$\text{score}_{\textit{signature-variant}} = 10 - \text{avg}\left(\text{n} + \text{n} + \text{n} + \text{n} + \text{n}\right)$$

respectively

$$\text{score}_{\textit{encryption-variant}} = 10 - \text{avg}\left(\text{n} + \text{n} + \text{n} + \text{n} + \text{n}\right)$$

and where $1 \leq \text{categories}_{\textit{algorithm}} \leq 5$ describes the number of different security categories offered by *algorithm*. We define

$$\text{general}_{\textit{algorithm}} = \begin{cases} 0 & \text{if } \textit{algorithm} \text{ is a general purpose algorithm} \\ 2 & \text{else} \end{cases}$$

to take into account if the algorithm is suitable for general use. Finally, by assigning numeric values of 0 = L = L, 1 = M = M and 2 = H = H, we define

$$\text{impl}_{\textit{algorithm}} = \text{avg}\left(\text{L} + \text{M} + \text{H} + \text{L} + \text{M} + \text{H}\right)$$

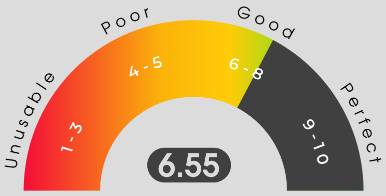
TBD: In the algorithm scores given in ID cards, we currently use $\text{impl}_{\textit{algorithm}} = 0$ in the respective computations because the necessary values for implementation complexity and size are still TBD. This will be corrected later.

Example: ML-DSA Overall Usability Score

We calculate

$$\text{score}_{\textit{ML-DSA-44}} = 10 - \text{avg}\left(3 + 3 + 2 + 1 + 5\right) = 10 - 2.8 = 7.2$$

Similarly, we obtain $\text{score}_{\textit{ML-DSA-65}} = 7.0$ and $\text{score}_{\textit{ML-DSA-87}} = 6.2$. Furthermore, $\text{categories}_{\textit{ML-DSA}} = 3$ since ML-DSA offers the three security categories II, III, and V, and $\text{general}_{\textit{ML-DSA}} = 0$ since ML-DSA is a general purpose signature algorithm. This results in an overall usability score of 6.55:



ML-DSA Overall Usability Score

$$\begin{aligned}
\text{score}_{\textit{ML-DSA}} &= \max\left\{0;\ \text{avg}\{\text{score}_{\textit{ML-DSA-44}}, \text{score}_{\textit{ML-DSA-65}}, \text{score}_{\textit{ML-DSA-87}}\} - \frac{1}{8}\cdot\left(5 - \text{categories}_{\textit{ML-DSA}}\right) - \text{general}_{\textit{ML-DSA}} - \frac{1}{8}\cdot\text{impl}_{\textit{ML-DSA}}\right\} \\
&= \max\left\{0;\ \text{avg}\{7.2;\ 7.0;\ 6.2\} - \frac{1}{8}\cdot(5-3) - 0 - 0\right\} \\
&= \max\{0;\ 6.8 - 0.25\} \\
&= \max\{0;\ 6.55\} \\
&= 6.55
\end{aligned}$$

TBD: We use $\text{impl}_{\textit{ML-DSA}} = 0$ because the necessary values to compute it are still TBD, cf. ML-DSA ID card. This will be corrected later.