Post-Quantum Cryptography Algorithm Cheat Sheets

Symbols

General color coding

Best to Worst: 🟩 🟨 🟧 🟥 🟥 TBD: 🟦

Symbols

❗ Recommendations differ depending on organization

⚙ Parameter set

🔒 Encryption

✏ Signing

📅 Not yet standardized by NIST

</> Implementation Code

Ⓢ Implementation size

🔑 Key Generation

🔓 Decryption

❓ Verification

▦ CPU Cycles

Ⓒ Implementation complexity

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 Work With This Cheat Sheet

TBD

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.).

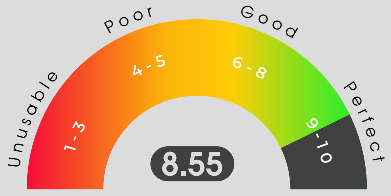
Pure vs. Pre-Hashing

- **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.

Post-Quantum Cryptography Algorithm Cheat Sheets

✎ Signature Algorithm Overview & ID Cards

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



Algorithm Overall Usability Score

Version	OID	Security Category	Performance 🔑	Performance ✎	Performance ?	Signature Size	Public Key Size	Suitable Pre-Hashing
ML-DSA-44	2.16.840.1.101.3.4.3.17	II	2	2	2	<1	<1	SHA-256, SHA3-256
ML-DSA-65	2.16.840.1.101.3.4.3.18	III	2	2	2	<1	<1	SHA-384, SHA3-384
ML-DSA-87	2.16.840.1.101.3.4.3.19	V	2	2	2	<1	<1	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	🔑	✎	?
Complexity	?	?	?
Size	?	?	?

👍 / Pros / Use If:

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

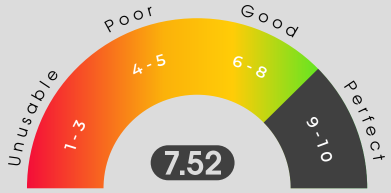
👎 / Cons / Don't Use If:

- 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.

No responsibility is taken for the correctness of this information.

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



Algorithm Overall Usability Score

Version	OID	Security Category	Performance			Signature Size	Public Key Size	Suitable Pre-Hashing
			🔑	✏	?			
Falcon-512	TBD	I	6	3	1	<1	<1	TBD
Falcon-1024	TBD	V	6	3	2	<1	<1	TBD

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

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

/ Pros / Use If:

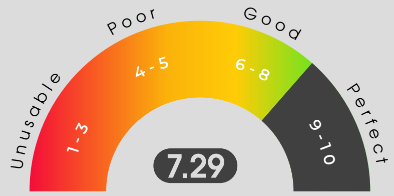
- Falcon has even smaller values for signature and public key sizes than ML-DSA, even though the values are in the same order of magnitude (<1 , <1 for both algorithms)

/ 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

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	?	?	?

Version	OID	Security Category	Performance			Signature Size	Public Key Size	Suitable Pre-Hashing
			🔑	✎	?			
SLH-DSA-SHA2-128s	2.16.840.1.101.3.4.3.20	I	5	6	3	<1	<1	SHA-256, SHA3-256
SLH-DSA-SHA2-128f	2.16.840.1.101.3.4.3.21	I	3	5	4	1	<1	SHA-256, SHA3-256
SLH-DSA-SHA2-192s	2.16.840.1.101.3.4.3.22	III	5	6	3	1	<1	SHA-384, SHA3-384
SLH-DSA-SHA2-192f	2.16.840.1.101.3.4.3.23	III	3	5	4	1	<1	SHA-384, SHA3-384
SLH-DSA-SHA2-256s	2.16.840.1.101.3.4.3.24	V	5	6	4	1	<1	SHA-512, SHA3-512
SLH-DSA-SHA2-256f	2.16.840.1.101.3.4.3.25	V	4	5	4	1	<1	SHA-512, SHA3-512
SLH-DSA-SHAKE-128s	2.16.840.1.101.3.4.3.26	I	5	6	3	<1	<1	SHA-256, SHA3-256
SLH-DSA-SHAKE-128f	2.16.840.1.101.3.4.3.27	I	3	5	4	1	<1	SHA-256, SHA3-256
SLH-DSA-SHAKE-192s	2.16.840.1.101.3.4.3.28	III	5	6	3	1	<1	SHA-384, SHA3-384
SLH-DSA-SHAKE-192f	2.16.840.1.101.3.4.3.29	III	3	5	4	1	<1	SHA-384, SHA3-384
SLH-DSA-SHAKE-256s	2.16.840.1.101.3.4.3.30	V	5	6	4	1	<1	SHA-512, SHA3-512
SLH-DSA-SHAKE-256f	2.16.840.1.101.3.4.3.31	V	4	5	4	1	<1	SHA-512, SHA3-512

👍 / Pros / Use If:

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

👎 / Cons / Don't Use If:

- Poor 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)

Unusable
Poor
Good
Perfect
1-3
4-5
6-8
9-10
0

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	?	?	?

Version	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

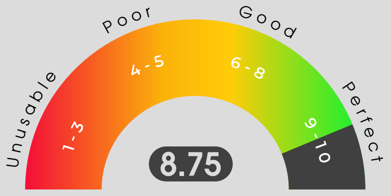
Note: Cycle counts for key generation, signing and verification depend on the CPU used. Values may vary on different CPUs and thus only a rough indicator.

LMS (Leighton-Micali Signature)

TBD

🔒 Encryption Algorithm Overview & ID Cards

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



Algorithm Overall Usability Score

Version	OID	Security Category	Performance			Ciphertext Size	Public Key Size
			🔑	🔒	🔓		
ML-KEM-512	2.16.840.1.101.3.4.4.7	I	1	1	1	<1	<1
ML-KEM-768	2.16.840.1.101.3.4.4.2	III	2	2	2	<1	<1
ML-KEM-1024	2.16.840.1.101.3.4.4.3	V	2	2	2	<1	<1

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

Algorithm Implementation	🔑	✎	❓
Complexity	?	?	?
Size	?	?	?

👍 / Pros / Use If:

- Currently the only post-quantum 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.

ℹ Algorithm Scoring: Algorithm Overall Usability Score

We try to measure an algorithm's overall usability by calculating a single number between 0 (worst) and 10 (best), taking into account all performance and size metrics, the number of security categories provided and whether or not is it is suitable for general use. An algorithm's overall score is computed as

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

where $\text{score}_{\textit{variant}}$ is a score for an individual algorithm variant 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 $1 \leq \gamma_{\textit{algorithm}} \leq 5$ describes the number of different security categories offered by *algorithm*. Finally, $\delta_{\textit{algorithm}} = \begin{cases} 0 & \text{if } \textit{algorithm} \text{ is a general purpose algorithm} \\ 2 & \text{else} \end{cases}$ takes into account if the algorithm is suitable for general use.

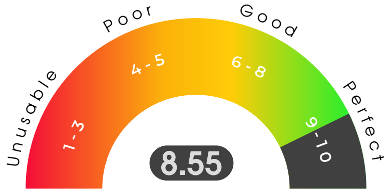
TBD: Take into account implementation complexity and size.

ℹ Example: ML-DSA

We calculate

$$\text{score}_{\textit{ML-DSA-44}} = \text{score}_{\textit{ML-DSA-65}} = \text{score}_{\textit{ML-DSA-87}} = 10 - \text{avg}\left(2 + 2 + 2 + 0 + 0\right) = 10 - 1.2 = 8.8$$

Furthermore, $\gamma_{\textit{ML-DSA}} = 3$ since ML-DSA offers the three security categories II, III, and V, and $\delta_{\textit{ML-DSA}} = 0$ since ML-DSA is a general purpose signature algorithm. This results in an overall usability score of 8.55:



Algorithm Overall Usability Score

$$\begin{aligned}
\text{score}_{\textit{ML-DSA}} &= \max\left\{(0;\ \text{avg}\{\text{score}_{\textit{ML-DSA-44}}, \text{score}_{\textit{ML-DSA-65}}, \text{score}_{\textit{ML-DSA-87}}\} - \frac{1}{8}\cdot\left(5-\gamma_{\textit{ML-DSA}}\right) - \delta_{\textit{ML-DSA}}\right\} \\
&= \max\left\{0;\ \text{avg}\{8.8;\ 8.8;\ 8.8\} - \frac{1}{8}\cdot(5-3) - 0\right\} \\
&= \max\{0;\ 8.8 - 0.25 - 0\} \\
&= \max\{0;\ 8.55\} \\
&= 8.55
\end{aligned}$$