THE POST-QUANTUM CRYPTOGRAPHY ALGORITHM CHEAT SHEETS

Mario Schiener

24 September 2024
v0.4 Draft

Symbols

General color coding

Best to Worst: ■ ■ ■ ■ ■ Links and TBDs: ■

Symbols

- Recommendations differ depending on organization
- Parameter set
- Key / Key Generation
- Encryption / Ciphertext
- Decryption
- Signing / Signature
- 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 is 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 The Cheat Sheets

The goal of this series of cheat sheets 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 sheets are 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 security or 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 of magnitude of base 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.

Algorithm Choice Guidance: Rules Of Thumb TBV

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

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

Objective: Performance of Key Generation

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

Objective: Performance of Signing

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

Objective: Performance of Verification

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

Objective: Small Signatures

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

Objective: Small Public Keys

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

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

Objective: Performance of Key Generation

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

Objective: Performance of Encryption

C. McEliece* | ML-KEM | BIKE | HQC | Frodo-KEM

Objective: Performance of Decryption

ML-KEM | C. McEliece* | HQC | BIKE | Frodo-KEM

Objective: Small Ciphertext

C. McEliece* | ML-KEM | BIKE | HQC | Frodo-KEM

Objective: Small Public Keys

ML-KEM | BIKE | HQC | Frodo-KEM | C. McEliece*

* Classic McEliece is only suitable for general encryption with limitations

Security Category Choices

- As a baseline, first consider using III parameters.
- 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.).
- Comparing post-quantum security categories of algorithms to their traditional security level in bits is a complex subject and like comparing apples and oranges to some degree.

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 apply:**
  - The message *M* is too large to be sent to the 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.

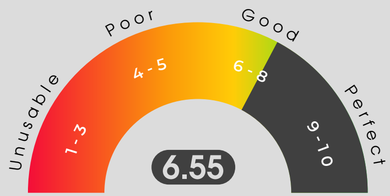
Pure PQC vs PQ/T Hybrid

This topic depends on too many factors (e.g. cost of migration, security considerations, risk profile, GRC requirements) to give general advice. Those aspects will differ greatly between different organizations. The main reasons to adopt PQ/T hybrids are to still have traditional algorithms in place in case a new algorithm turns out to be insecure, and that PQ/T might help to avoid a big bang migration as systems could simply ignore the PQC component if they do not support it yet. Consider recommendations from different government agencies: BSI and ANSSI recommend PQ/T hybrid strategies, whereas NIST is more reserved towards PQ/T. **If using PQ/T hybrids, preferably use ECC (e.g. secp256r1, brainpoolP256r1, Curve25519)** instead of RSA for the traditional component to keep key sizes as small as possible.

SIGNATURE ALGORITHMS

ID Cards

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



Algorithm Overall Usability Score

Parameter Set	OID	Security Category	Performance (keygen)	Performance (sign)	Performance (verify)	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
Suitable For General Use	Yes
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)

Implementation	Keygen	Sign	Verify
Complexity	?	?	?
Size	?	?	?

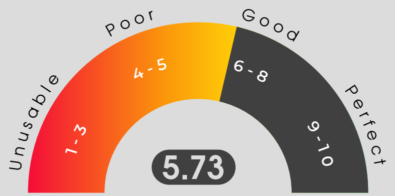
/ Pros / Use If:

- You need a general-purpose signature algorithm with decent specs in all categories
- Usually the best option for a signature algorithm in most protocols (e.g. TLS, SSH, S/MIME, etc.).
- Good choice for X.509 certificates, including CA certificates.

/ Cons / Don't Use If:

- You don't want a lattice-based algorithm

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



Algorithm Overall Usability Score

Previous Name	Falcon
Specification	Project Page
Type	Signature
Family	Lattice
Suitable For General Use	Yes
Standardization Status	Pending
Recommended By	TBD
Hashing	TBD
Naming	TBD

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

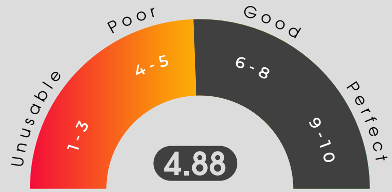
/ Pros / Use If:

- Generally good for use in most protocols (e.g. TLS, SSH, S/MIME, etc.). Exceptions may apply in cases of repeated generation of ephemeral keys (poor performance).
- 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
- Less suitable in applications involving the creation of lots of ephemeral keys/certificates (key generation is slow).

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



Algorithm Overall Usability Score

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

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

Parameter Set	OID	Security Category	Performance: Key generation	Performance: Signing	Performance: Verification	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
- Generally good for use in most protocols, even if ML-DSA is the preferred option.
- Use as a more conservative choice in use cases with less frequent algorithm execution, i.e. where latency or message size are not a big concern. For example, S/MIME, document or code signing, creation of CAs.

/ 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

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)
Suitable For General Use	No
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

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

Parameter Set	Numeric Identifier	Security Category	Performance (Key generation)	Performance (Signing)	Performance (Verification)	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

LMS/HSS (Leighton-Micali Signature / Hierarchical Signature System)

Algorithm Overall Usability Score

Previous Name	LMS, LMS/HSS
Specification	SP 800-208, RFC 8554
Type	Signature
Family	Merkle Trees (stateful hash trees)
Suitable For General Use	No
Standardization Status	Standardized
Recommended By	NIST, BSI, ANSSI
Hashing	TBD
Naming	LMS_SHA256_M32_[h] where h is the tree height

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

Parameter Set	**Numeric Identifier**	**Security Category**	**Performance** (Key generation)	**Performance** (Signing)	**Performance** (Verification)	**Signature Size**	**Maximum Signatures**
LMS_SHA256_M32_H5	0x00000005	?	?	?	?	?	2^{5}
LMS_SHA256_M32_H10	0x00000006	?	?	?	?	?	2^{10}
LMS_SHA256_M32_H15	0x00000007	?	?	?	?	?	2^{15}
LMS_SHA256_M32_H20	0x00000008	?	?	?	?	?	2^{20}
LMS_SHA256_M32_H25	0x00000009	?	?	?	?	?	2^{25}

Note:
SP 800-208 defines further parameter sets not listed in RFC 8554 using other hash functions (SHA256/192, SHAKE256/256, SHAKE256/192). 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. Similar to XMSS multi-tree variants, a comparable structure exists with Hierarchical Signature System (HSS) variants of LMS that involves a number of LMS instances.

/ 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

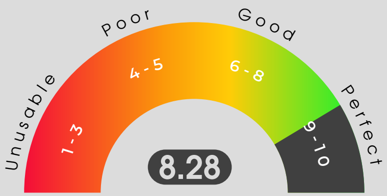
/ Cons / Don't Use If:

- 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

ENCRYPTION / KEM ALGORITHMS

ID Cards

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



Algorithm Overall Usability Score

Parameter Set	OID	Security Category	Performance (key generation)	Performance (encapsulation)	Performance (decapsulation)	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
Suitable For General Use	Yes
Standardization Status	Standardized
Recommended By	NIST, BSI, ANSSI
Naming	TBD

Implementation	Key generation	Encapsulation	Decapsulation
Complexity	?	?	?
Size	?	?	?

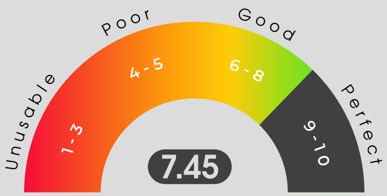
/ Pros / Use If:

- 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
- Usually the best choice in most protocols involving an encryption/KEM component (e.g. TLS, S/MIME, etc.)

/ Cons / Don't Use If:

- You don't want a lattice-based algorithm

BIKE (Bit Flipping Key Encapsulation)



Algorithm Overall Usability Score

Parameter Set	OID	Security Category	Performance (keygen)	Performance (encaps)	Performance (decaps)	Ciphertext Size	Public Key Size
BIKE-L1	TBD	I	3	2	4	0	0
BIKE-L3	TBD	III	4	3	5	1	1
BIKE-L5	TBD	V	n/a	n/a	n/a	2	2

Previous Name BIKE
Specification Project Page
Type Encryption/KEM
Family Codes
Suitable For General Use Yes
Standardization Status 4th round candidate
Recommended By TBD
Naming after security categories I, III and V

Implementation	Key	Edit	?
Complexity	?	?	?
Size	?	?	?

Note:
No data available for BIKE-L5 for cycle counts. The algorithm overall usability score is computed over BIKE-L1 and BIKE-L3 only.

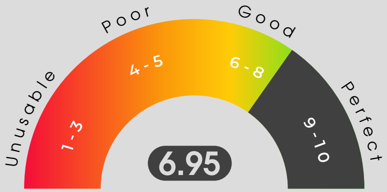
/ Pros / Use If:

- Possible alternative to ML-KEM not based on lattices.

/ Cons / Don't Use If:

- Not yet standardized. Note that NIST intends to standardize at most one of the algorithms HQC and BIKE.
- Metrics not as good as the ones of ML-KEM.

HQC (Hamming-Quasi-Cyclic)



Algorithm Overall Usability Score

Previous Name	HQC
Specification	Project Page
Type	Encryption/KEM
Family	Codes
Suitable For General Use	Yes
Standardization Status	4th round candidate
Recommended By	TBD
Naming	128, 192 and 256 bits of security in reference to security categories I, III and V

Implementation			
Complexity	?	?	?
Size	?	?	?

Parameter Set	OID	Security Category	Performance			Ciphertext Size	Public Key Size
HQC-128	TBD	I	2	3	3	2	1
HQC-192	TBD	III	3	3	4	3	2
HQC-256	TBD	V	3	4	4	3	2

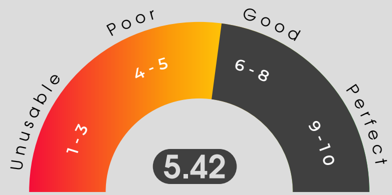
/ Pros / Use If:

- Possible alternative to ML-KEM not based on lattices.

/ Cons / Don't Use If:

- Not yet standardized. Note that NIST intends to standardize at most one of the algorithms HQC and BIKE.
- Metrics not as good as the ones of ML-KEM.

FrodoKEM



Algorithm Overall Usability Score

Previous Name	FrodoKEM
Specification	Project Page
Type	Encryption/KEM
Family	Lattice
Suitable For General Use	Yes
Standardization Status	Disregarded by NIST
Recommended By	BSI, ANSSI
Naming	FrodoKEM-[n]-[AES | SHAKE] where n is a matrix dimension

Implementation			
Complexity	?	?	?
Size	?	?	?

Parameter Set	**OID**	**Security Category**	**Performance**			**Ciphertext Size**	**Public Key Size**
FrodoKEM-640-AES	TBD	I	4	4	4	3	3
FrodoKEM-640-SHAKE	TBD	I	5	5	5	3	3
FrodoKEM-976-AES	TBD	III	4	4	4	3	3
FrodoKEM-976-SHAKE	TBD	III	6	6	6	3	3
FrodoKEM-1344-AES	TBD	V	5	5	5	4	4
FrodoKEM-1344-SHAKE	TBD	V	6	6	6	4	4

Note:
There are also ephemeral versions of FrodoKEM (eFrodoKEM-640, etc.), but BSI only recommends FrodoKEM-976 and FrodoKEM-1344 versions for long-term protection. The algorithm score also only includes the non-ephemeral versions listed above.

/ Pros / Use If:

- More conservative than ML-KEM since it is based on unstructured grids

/ Cons / Don't Use If:

- Not standardized by NIST because ML-KEM is more efficient.

Classic McEliece

Algorithm Overall Usability Score

Previous Name Classic McEliece
Specification Project Page
Type Encryption/KEM
Family Codes
Suitable For General Use No / With Limitations
Standardization Status 4th round candidate (NIST)
Recommended By BSI
Naming mceliece[n][deg(F(y))][[f]]
where n, F(y) are parameters
non-f versions: $(\mu, \nu) = (0, 0)$
f versions: $(\mu, \nu) = (32, 64)$

Implementation			
Complexity	?	?	?
Size	?	?	?

Parameter Set	OID	Security Category	Performance (keygen)	Performance (encaps)	Performance (decaps)	Ciphertext Size	Public Key Size
mceliece348864	TBD	I	6	2	3	0	8
mceliece348864f	TBD	I	6	n/a	n/a	0	8
mceliece460896	TBD	III	7	2	3	0	9
mceliece460896f	TBD	III	7	n/a	n/a	0	9
mceliece6688128	TBD	V	8	3	3	0	10
mceliece6688128f	TBD	V	7	n/a	n/a	0	10
mceliece6960119	TBD	V	7	3	3	0	10
mceliece6960119f	TBD	V	7	n/a	n/a	0	10
mceliece8192128	TBD	V	8	3	3	0	10
mceliece8192128f	TBD	V	8	n/a	n/a	0	10

Note:
No data available for the f versions for encryption/decryption cycle counts. The algorithm overall usability score is computed over non-f versions only. Non-f and f versions are interoperable, with f versions offering faster key generation and non-f versions having simpler key generation.

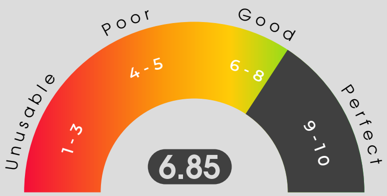
/ Pros / Use If:

- Possible alternative to ML-KEM not based on lattices.
- Considered safe and recommended by BSI even though not standardized by NIST.
- Use cases that allow for pre-sharing of keys where the large public key size is not a concern.

/ Cons / Don't Use If:

- Not yet standardized by NIST.
- Not suitable for general use due to the very large public keys and poor performance of key generation.
- Due to aforementioned point, likely not suitable for embedded devices.

NTRU (Number Theory Research Unit / Number Theorists 'R' Us)



Algorithm Overall Usability Score

Parameter Set	OID	Security Category	Performance (key generation)	Performance (encapsulation)	Performance (decapsulation)	Ciphertext Size	Public Key Size
NTRUHPS2048509	N/A	I	5	4	4	0	0
NTRUHRSS701	N/A	III	6	4	5	0	0
NTRUHPS2048677	N/A	III	6	4	5	0	0
NTRUHPS4096821	N/A	V	6	4	5	0	0

Previous Name	NTRU
Specification	Project Page
Type	Encryption/KEM
Family	Lattice
Suitable For General Use	Yes
Standardization Status	Not Standardized
Recommended By	n/a
Naming	author initials and a prime number

Implementation			
Complexity	?	?	?
Size	?	?	?

/ Pros / Use If:

- The intellectual property rights of ML-KEM have been unclear for a while, with NTRU being a possible backup alternative without such issues. This topic is now considered sufficiently clear and resolved by NIST. In case legal concerns about ML-KEM still apply in some exceptional cases, NTRU might be an option. However, ML-KEM is usually the better option in terms of technical aspects.

/ Cons / Don't Use If:

- Not standardized
- Performance not as good as ML-KEM, especially for key generation. Similarly, BIKE and HQC offer better metrics.

HASHING

Overview

Overview: Hash Algorithms and Extendable Output Functions (XOF) TBV

Family	Variant	Output Size [Bits]	Rounds	Construction	Traditional Pre-Quantum Security [Bits] Collision	Pre-Image	2nd Pre-Image	Post-Quantum Security [Bits] Collision	Pre-Image	2nd Pre-Image	Performance
SHA-2	SHA-224	224	64	Merkle–Damgård construction	112	224	224	74	112	112	7.62
	SHA-256	256	64	Merkle–Damgård construction	128	256	256	85	128	128	7.63
	SHA-384	384	80	Merkle–Damgård construction	192	384	384	128	192	192	5.12
	SHA-512	512	80	Merkle–Damgård construction	256	512	512	170	256	256	5.06
	SHA-512/224	224	80	Merkle–Damgård construction	112	224	224	TBD	TBD	TBD	≈ SHA-384
	SHA-512/256	256	80	Merkle–Damgård construction	128	256	256	TBD	TBD	TBD	≈ SHA-384
SHA-3	SHA3-224	224	24	Keccak sponge construction	112	224	224	74	112	112	8.12
	SHA3-256	256	24	Keccak sponge construction	128	256	256	85	128	128	8.59
	SHA3-384	384	24	Keccak sponge construction	192	384	384	128	192	192	11.06
	SHA3-512	512	24	Keccak sponge construction	256	512	512	170	256	256	15.88
	SHAKE128	d (arbitrary)	24	Keccak sponge construction	$\min\left(\frac{d}{2}; 128\right)$	$\min\left(\frac{d}{2}; 128\right)$	$\min\left(\frac{d}{2}; 128\right)$	$\min\left(\frac{d}{3}; 128\right)$	$\min\left(\frac{d}{2}; 128\right)$	$\min\left(\frac{d}{2}; 128\right)$	7.08
	SHAKE256	d (arbitrary)	24	Keccak sponge construction	$\min\left(\frac{d}{2}; 256\right)$	$\min\left(\frac{d}{2}; 256\right)$	$\min\left(\frac{d}{2}; 256\right)$	$\min\left(\frac{d}{3}; 256\right)$	$\min\left(\frac{d}{2}; 256\right)$	$\min\left(\frac{d}{2}; 256\right)$	8.59

Notes & Takeaways

- SHAKE128 and SHAKE256 are exendable output functions and allow for an arbitrary output size d.
- SHA-512/224 and SHA-512/256 denotes the truncation of the SHA-512 hash to 224 resp. 256 bits.
- Security against collision attacks is specified using values according to Brassard et al. (Source)
- Performance is measured in median cycles/byte on a Skylake CPU (Source).
- Hash functions other than SHA-2 and SHA-3 variants (e.g. SHA-1, RipeMD-160, MD5, etc.) should generally be avoided as they are less secure. They are therefore not part of the above overview.

APPENDIX

Traditional vs. Post-Quantum Security Level Comparison TBV

Traditional Pre-Quantum Security: Security [Bits]	Classical Attack Complexity	Attack in terms of bit size n	Algorithm Examples	NIST Security Categories defined through attack	Post-Quantum Security: Classical Attack Improvement	Quantum Attack Complexity	Security [Bits]	Signature Parameter Sets	Encryption/KEM Parameter Sets
n	$\mathcal{O}\left(2^n\right)$	n-bit key space exhaustive key search	AES		Grover	$\mathcal{O}\left(2^{\frac{n}{2}}\right)$	$\frac{n}{2}$		
n	$\mathcal{O}\left(2^n\right)$	n-bit hash pre-image attack	SHA-2, SHA-3		Grover	$\mathcal{O}\left(2^{\frac{n}{2}}\right)$	$\frac{n}{2}$		
n	$\mathcal{O}\left(2^n\right)$	n-bit hash 2nd-pre-image attack	SHA-2, SHA-3		Grover	$\mathcal{O}\left(2^{\frac{n}{2}}\right)$	$\frac{n}{2}$		
$\frac{n}{2}$	$\mathcal{O}\left(2^{\frac{n}{2}}\right)$	n-bit hash collision attack	SHA-2, SHA-3		Brassard et al.	$\mathcal{O}\left(2^{\frac{n}{3}}\right)$	$\frac{n}{3}$		
128	$\mathcal{O}\left(2^{128}\right)$	128-bit key space exhaustive key search	AES-128	Category 1	Grover	$\mathcal{O}\left(2^{64}\right)$	64	Falcon-512, SLH-DSA 128 variants	ML-KEM-512, BIKE-L1, HQC-128, FrodoKEM-640
		128-bit hash pre-image attack	N/A		Grover	$\mathcal{O}\left(2^{64}\right)$	64		
		128-bit hash 2nd-pre-image attack	N/A		Grover	$\mathcal{O}\left(2^{64}\right)$	64		
		256-bit hash collision attack	SHA-256, SHA3-256	Category 2	Brassard et al.	$\mathcal{O}\left(2^{85}\right)$	85	ML-DSA-44, TBV LMS	
192	$\mathcal{O}\left(2^{192}\right)$	192-bit key space exhaustive key search	AES-192	Category 3	Grover	$\mathcal{O}\left(2^{96}\right)$	96	ML-DSA-67, SLH-DSA 192 variants	ML-KEM-768, BIKE-L3, HQC-192, FrodoKEM-976
		192-bit hash pre-image attack	N/A		Grover	$\mathcal{O}\left(2^{96}\right)$	96		
		192-bit hash 2nd-pre-image attack	N/A		Grover	$\mathcal{O}\left(2^{96}\right)$	96		
		384-bit hash collision attack	SHA-384, SHA3-384	Category 4	Brassard et al.	$\mathcal{O}\left(2^{128}\right)$	128		
256	$\mathcal{O}\left(2^{256}\right)$	256-bit key space exhaustive key search	AES-256	Category 5	Grover	$\mathcal{O}\left(2^{128}\right)$	128	ML-DSA-85, SLH-DSA 256 variants TBV XMS	ML-KEM-1024, BIKE-L5, HQC-256, FrodoKEM-1344
		256-bit hash pre-image attack	N/A		Grover	$\mathcal{O}\left(2^{128}\right)$	128		
		256-bit hash 2nd-pre-image attack	N/A		Grover	$\mathcal{O}\left(2^{128}\right)$	128		
		512-bit hash collision attack	SHA-512, SHA3-512		Brassard et al.	$\mathcal{O}\left(2^{170}\right)$	170		
384	$\mathcal{O}\left(2^{384}\right)$	384-bit key space exhaustive key search	N/A		Grover	$\mathcal{O}\left(2^{192}\right)$	192		
		384-bit hash pre-image attack	SHA-384, SHA3-384		Grover	$\mathcal{O}\left(2^{192}\right)$	192		
		384-bit hash 2nd-pre-image attack	SHA-384, SHA3-384		Grover	$\mathcal{O}\left(2^{192}\right)$	192		
		768-bit hash collision attack	N/A		Brassard et al.	$\mathcal{O}\left(2^{256}\right)$	256		
512	$\mathcal{O}\left(2^{512}\right)$	512-bit key space exhaustive key search	N/A		Grover	$\mathcal{O}\left(2^{256}\right)$	256		
		512-bit hash pre-image attack	SHA-512, SHA3-512		Grover	$\mathcal{O}\left(2^{256}\right)$	256		
		512-bit hash 2nd-pre-image attack	SHA-512, SHA3-512		Grover	$\mathcal{O}\left(2^{256}\right)$	256		
		1024-bit hash collision attack	N/A		Brassard et al.	$\mathcal{O}\left(2^{341}\right)$	341		

Notes & Takeaways

- The above is still a simplified view. It does not dive into the specific assumptions made in the new algorithms' security proofs and the possible attacks on those algorithms (e.g., is a collision attack on a hash function enough, or would a pre-image be required to break the signature scheme using that hash function). It is also not including considering practical considerations and predictions about quantum computers, such as runtime or quantum circuit depth. Cf. NIST PQC Evaluation Criteria, 4.A.5 for further details.
- A pre-quantum security level can have different post-quantum security levels equivalents. It all depends on the specific attack in question, resp. the security assumption a new post-quantum algorithm is based on or must be secure against. For hashing, the often cited "SHA-256 offers 128 bits of security" is not wrong, but only part of the picture: it offers 256 bits of security against pre-image and 2nd-preimage-attacks. The n bits of security against 2nd-preimage attacks is still a slightly simplified and harmonized view to avoid different formulas for different hash functions. Cf. NIST: Hash Functions for details.
- The fact that the exact level of post-quantum security still depends on so many assumptions about CRQCs, comparing pre-quantum and post-quantum security levels remains an apples and oranges comparison to some degree. Such comparisons should be taken with a grain of salt, especially when done only on a rather superficial level.
- The lines marked with colors ■, ■, ■, ■ and ■ are the ones used by NIST to define categories I, II, III, IV and V. The lines marked in lighter colors ■, ■, and ■ have the same post-quantum security as the defining lines of categories I, III and V, respectively. They therefore fulfill the "comparable to or greater than" clause in NIST's category definitions. Note that categories IV and V are differentiated despite having 128 bits of post-quantum security – again highlighting the fact that a precise categorization is difficult and requires a detailed analysis of many aspects.

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 parameter sets, 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

$$\mathbf{score}_{algorithm} = \mathbf{max}\left\{0,\ \mathbf{avg}\left\{\mathbf{score}_{parameterSet} \mid \mathbf{\mathit{parameterSet}}\ \mathbf{is\ a\ parameter\ set\ of}\ \mathbf{\mathit{algorithm}}\right\} - \frac{1}{8}\cdot\left(5 - \mathbf{categories}_{algorithm}\right) - \frac{1}{4}\cdot\mathbf{impl}_{algorithm} - \mathbf{generality}_{algorithm}\right\}$$

where $\text{score}_{parameterSet}$ is a score for an individual algorithm parameter set. Depending on the type of algorithm, $\text{score}_{parameterSet}$ is defined as

$$\text{score}_{signature\text{-}parameterSet} = 10 - \text{avg}\left\{n, n, n, n, n\right\} \qquad \text{respectively} \qquad \text{score}_{encryption\text{-}parameterSet} = 10 - \text{avg}\left\{n, n, n, n, n\right\}.$$

Furthermore, $1 \leq \text{categories}_{algorithm} \leq 5$ denotes the number of different NIST security categories offered by *algorithm*. By assigning numeric values of 0 = L = L, 1 = M = M and 2 = H = H, we set

$$\text{impl}_{algorithm} = \text{avg}\left\{L, M, H, L, M, H\right\}.$$

Finally, we define

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

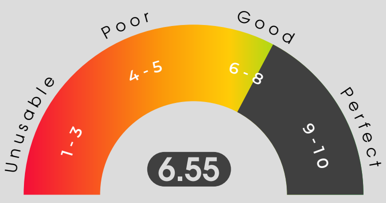
to take into account if the algorithm is suitable for general use.
TBD: In the algorithm scores given in ID cards, we currently use $\text{impl}_{algorithm}$ = 0 in the respective computations because the necessary values for implementation complexity and size are still TBD. This will be corrected later.

Example: ML-DSA Overall Usability Score

We calculate

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

Similarly, we obtain $\text{score}_{ML\text{-}DSA\text{-}65}$ = 7.0 and $\text{score}_{ML\text{-}DSA\text{-}87}$ = 6.2. Furthermore, $\text{categories}_{ML\text{-}DSA}$ = 3 since ML-DSA offers the three security categories II, III, and V, and $\text{generality}_{ML\text{-}DSA}$ = 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

$$\begin{aligned}
\text{score}_{ML\text{-}DSA} &= \max\left\{0,\ \text{avg}\left\{\text{score}_{ML\text{-}DSA\text{-}44}, \text{score}_{ML\text{-}DSA\text{-}65}, \text{score}_{ML\text{-}DSA\text{-}87}\right\} - \frac{1}{8}\cdot\left(5 - \text{categories}_{ML\text{-}DSA}\right) - \frac{1}{4}\cdot\text{impl}_{ML\text{-}DSA} - \text{generality}_{ML\text{-}DSA}\right\} \\
&= \max\left\{0,\ \text{avg}\left\{7.2, 7.0, 6.2\right\} - \frac{1}{8}\cdot(5\text{-}3) - 0 - 0\right\} \\
&= \max\left\{0,\ 6.8 - 0.25\right\} \\
&= \max\left\{0,\ 6.55\right\} \\
&= 6.55
\end{aligned}$$

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