Classic McEliece: conservative code-based cryptography

Daniel J. Bernstein, Tung Chou, Carlos Cid,
Jan Gilcher, Tanja Lange, Varun Maram,
Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen,
Edoardo Persichetti, Christiane Peters, Nicolas Sendrier,
Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, Wen Wang

https://classic.mceliece.org/

17 September 2024

Conservative

Classic McEliece is the paramount conservative code-based encryption scheme.

Conservative

Classic McEliece is the paramount conservative code-based encryption scheme.

— "Wait, how is Classic McEliece conservative when there are papers giving fast breaks of various security properties for McEliece-based systems? See, e.g., 1999 Hall–Goldberg–Schneier reaction attacks, 2021 Chou break of NTS-KEM, 2024 Bernstein–Lange break of PALOMA."

Conservative

Classic McEliece is the paramount conservative code-based encryption scheme.

— "Wait, how is Classic McEliece conservative when there are papers giving fast breaks of various security properties for McEliece-based systems? See, e.g., 1999 Hall–Goldberg–Schneier reaction attacks, 2021 Chou break of NTS-KEM, 2024 Bernstein–Lange break of PALOMA."

— Classic McEliece is carefully designed to rely on a minimal, well-studied security assumption for McEliece: OW-CPA, i.e., OW-Passive, i.e., hardness of the pure search problem for a random plaintext.

Careful security analysis

Top-down view of the analysis in the Classic McEliece security guide:

- Security goal: **IND-CCA2 KEM**. (See Section 1. Use separate modules for generic transformations beyond IND-CCA2 KEM; see Section 6.)

Careful security analysis

Top-down view of the analysis in the Classic McEliece security guide:

- Security goal: **IND-CCA2 KEM**. (See Section 1. Use separate modules for generic transformations beyond IND-CCA2 KEM; see Section 6.)
- Selected hash function: SHAKE256. Focus on **QROM IND-CCA2**. (See Section 5.3.3.)

Careful security analysis

Top-down view of the analysis in the Classic McEliece security guide:

- Security goal: **IND-CCA2 KEM**. (See Section 1. Use separate modules for generic transformations beyond IND-CCA2 KEM; see Section 6.)
- Selected hash function: SHAKE256. Focus on **QROM IND-CCA2**. (See Section 5.3.3.)
- QROM IND-CCA2 for Classic McEliece follows tightly from **OW-CPA security of underlying PKE**. (See Section 5.)

Careful security analysis

Top-down view of the analysis in the Classic McEliece security guide:

- Security goal: **IND-CCA2 KEM**. (See Section 1. Use separate modules for generic transformations beyond IND-CCA2 KEM; see Section 6.)
- Selected hash function: SHAKE256. Focus on **QROM IND-CCA2**. (See Section 5.3.3.)
- QROM IND-CCA2 for Classic McEliece follows tightly from **OW-CPA security of underlying PKE**. (See Section 5.)
- OW-CPA security of this PKE follows tightly from **OW-CPA security of original McEliece PKE**. (See Section 4.)

Careful security analysis

Top-down view of the analysis in the Classic McEliece security guide:

- Security goal: **IND-CCA2 KEM**. (See Section 1. Use separate modules for generic transformations beyond IND-CCA2 KEM; see Section 6.)
- Selected hash function: SHAKE256. Focus on **QROM IND-CCA2**. (See Section 5.3.3.)
- QROM IND-CCA2 for Classic McEliece follows tightly from **OW-CPA security of underlying PKE**. (See Section 5.)
- OW-CPA security of this PKE follows tightly from **OW-CPA security of original McEliece PKE**. (See Section 4.)
- Review then focuses on OW-CPA attacks. (See Section 3.)

Careful security analysis

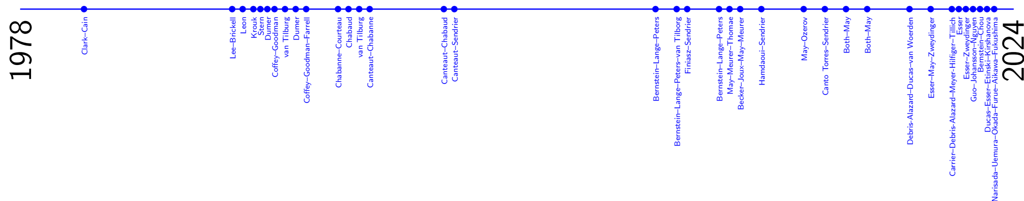
Top-down view of the analysis in the Classic McEliece security guide:

- Security goal: **IND-CCA2 KEM**. (See Section 1. Use separate modules for generic transformations beyond IND-CCA2 KEM; see Section 6.)
- Selected hash function: SHAKE256. Focus on **QROM IND-CCA2**. (See Section 5.3.3.)
- QROM IND-CCA2 for Classic McEliece follows tightly from **OW-CPA security of underlying PKE**. (See Section 5.)
- OW-CPA security of this PKE follows tightly from **OW-CPA security of original McEliece PKE**. (See Section 4.)
- Review then focuses on OW-CPA attacks. (See Section 3.)

The only ways something can possibly go wrong: disaster involving SHAKE256; mistake in tight reductions; better OW-CPA attack against original McEliece.

McEliece security stability

$$\lim_{K\to\infty} \frac{\log_2 \mathrm{AttackCost}_{\mathrm{year}}(K)}{\log_2 \mathrm{AttackCost}_{2024}(K)}$$



McEliece security stability

Blue: McEliece.

Red: Lattices had 45% higher security levels in 2010 than they have today.

$$\lim_{K\to\infty} \frac{\log_2 \text{AttackCost}_{\text{year}}(K)}{\log_2 \text{AttackCost}_{2024}(K)}$$

∞
1.453
1.344
1.180
1978
2024

Ajtai–Kumar–Sivakumar
Nguyen–Vidick
Micciancio–Voulgaris
Wang–Liu–Tian–Bi
Zhang–Pan–Hu
Laarhoven
Laarhoven–de Weger
Becker–Ducas–Gama–Laarhoven
Bernstein

Clark–Cain
Lee–Brickell
Leon
Krouk
Stern
Dumer
Coffey–Goodman
van Tilburg
Dumer
Coffey–Goodman–Farrell
Chabanne–Courteau
Chabaud
van Tilburg
Canteaut–Chabanne
Canteaut–Chabaud
Canteaut–Sendrier
Bernstein–Lange–Peters
Bernstein–Lange–Peters–van Tilborg
Finiasz–Sendrier
Bernstein–Lange–Peters
May–Meurer–Thomae
Becker–Joux–May–Meurer
Hamdaoui–Sendrier
May–Ozerov
Canto Torres–Sendrier
Both–May
Both–May
Debris-Alazard–Ducas–van Woerden
Esser–May–Zweydinger
Carrier–Debris-Alazard–Meyer-Hilfiger–Tillich
Esser
Esser–Zweydinger
Guo–Johansson–Nguyen
Bernstein–Chou
Ducas–Esser–Etinski–Kirshanova
Narisada–Uemura–Okada–Furue–Aikawa–Fukushima

McEliece security stability

$$\lim_{K\to\infty} \frac{\log_2 \text{AttackCost}_{\text{year}}(K)}{\log_2 \text{AttackCost}_{2024}(K)}$$

Blue: McEliece.

Red: Lattices had 45% higher security levels in 2010 than they have today.

Length of the history matters:
SIKE security was stable 2011–2022.

Measuring stability beyond asymptotics

Security guide, Section 3.5, "Concrete costs of information-set decoding": asymptotics say only what happens "as $n \to \infty$. More detailed attack-cost evaluation is therefore required for any particular parameters."

Measuring stability beyond asymptotics

Security guide, Section 3.5, "Concrete costs of information-set decoding": asymptotics say only what happens "as $n \to \infty$. More detailed attack-cost evaluation is therefore required for any particular parameters."

2023.10 Bernstein talk gave two non-asymptotic stability metrics:

- PQCrypto 2008 Bernstein–Lange–Peters attack software is as fast as Eurocrypt 2022 attack software for current challenges.

Measuring stability beyond asymptotics

Security guide, Section 3.5, "Concrete costs of information-set decoding": asymptotics say only what happens "as $n \to \infty$. More detailed attack-cost evaluation is therefore required for any particular parameters."

2023.10 Bernstein talk gave two non-asymptotic stability metrics:

- PQCrypto 2008 Bernstein–Lange–Peters attack software is as fast as Eurocrypt 2022 attack software for current challenges.
- Crypto 2024 Bernstein–Chou "CryptAttackTester: high-assurance attack analysis": software to (1) build complete attack circuits, (2) predict circuit cost and probability, (3) run small attacks to check accuracy. CryptAttackTester predicts $2^{156.96}$ bit ops for 348864 using attack ideas from the 1980s (`isd1`), $2^{150.59}$ bit ops using latest attacks (`isd2`).

Understanding a security-level comparison

Fix a ciphertext size. Known attacks against Classic McEliece then cost much more than known attacks against Kyber, NTRU, etc. Why does this happen?

Understanding a security-level comparison

Fix a ciphertext size. Known attacks against Classic McEliece then cost much more than known attacks against Kyber, NTRU, etc. Why does this happen?

See 2024.07 Bernstein talk for an answer. Outline:

- Attacking code-based ciphertexts, like attacking lattice-based ciphertexts, is equivalent to finding a lattice vector close to a target point.

Understanding a security-level comparison

Fix a ciphertext size. Known attacks against Classic McEliece then cost much more than known attacks against Kyber, NTRU, etc. Why does this happen?

See 2024.07 Bernstein talk for an answer. Outline:

- Attacking code-based ciphertexts, like attacking lattice-based ciphertexts, is equivalent to finding a lattice vector close to a target point.
- Direct quantitative comparison of keygen+dec decoding power: McEliece creates vectors much farther from lattice than Kyber, NTRU, etc.

Understanding a security-level comparison

Fix a ciphertext size. Known attacks against Classic McEliece then cost much more than known attacks against Kyber, NTRU, etc. Why does this happen?

See 2024.07 Bernstein talk for an answer. Outline:

- Attacking code-based ciphertexts, like attacking lattice-based ciphertexts, is equivalent to finding a lattice vector close to a target point.
- Direct quantitative comparison of keygen+dec decoding power: McEliece creates vectors much farther from lattice than Kyber, NTRU, etc.
- Conclusion about difficulty of attack problem: the bigger distance easily explains a big increase in attack cost.

Very large security margin against key recovery

Long history of papers on recovering McEliece private keys from public keys.
Sometimes faster than ISD for extreme parameter sets,
but always much slower than ISD for our selected parameter sets.
These attacks would need gigantic improvements to threaten our security targets.

Very large security margin against key recovery

Long history of papers on recovering McEliece private keys from public keys.
Sometimes faster than ISD for extreme parameter sets,
but always much slower than ISD for our selected parameter sets.
These attacks would need gigantic improvements to threaten our security targets.

Very different from the common situation of cryptosystems having
key-recovery problems as weak as message-recovery problems.

Very large security margin against key recovery

Long history of papers on recovering McEliece private keys from public keys.
Sometimes faster than ISD for extreme parameter sets,
but always much slower than ISD for our selected parameter sets.
These attacks would need gigantic improvements to threaten our security targets.

Very different from the common situation of cryptosystems having
key-recovery problems as weak as message-recovery problems.

There are also papers on merely *distinguishing* McEliece public keys from
random matrices. Again much slower than ISD for our selected parameter sets,
and structurally irrelevant to the Classic McEliece security analysis.

Software correctness, part 1

The official Classic McEliece software is designed for deployment.

Internally, this software is CHES 2017 Chou "McBits revisited"

+ extending software to more parameter sets
+ various speedups (e.g., recent enc speedup)
+ small tweaks for, e.g., private-key format.

Software correctness, part 1

The official Classic McEliece software is designed for deployment.
Internally, this software is CHES 2017 Chou "McBits revisited"
+ extending software to more parameter sets
+ various speedups (e.g., recent enc speedup)
+ small tweaks for, e.g., private-key format.

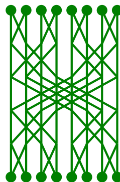
We also converted the specification into a Sage package.
Not designed for deployment; designed to maximize readability.
SUPERCOP checksums (known-answer tests, including modified ciphertexts) match between Sage package and official software for all parameter sets.

Software correctness, part 1

The official Classic McEliece software is designed for deployment.
Internally, this software is CHES 2017 Chou "McBits revisited"
+ extending software to more parameter sets
+ various speedups (e.g., recent enc speedup)
+ small tweaks for, e.g., private-key format.

We also converted the specification into a Sage package.
Not designed for deployment; designed to maximize readability.
SUPERCOP checksums (known-answer tests, including modified ciphertexts) match between Sage package and official software for all parameter sets.

Many eyeballs on spec. Every function implemented at least twice.
Only one component, control-bits computation for private key, had same implementor for software and Sage package; that component uses formulas with computer-checked proofs and is double-checked by much simpler code.



Software correctness, part 2

The formulas that we use for decoding now have computer-checked proofs of working correctly for all inputs.

Interesting spinoff, unusual advantage of this cryptosystem: this decoding algorithm is rigid even without reencryption.

Software correctness, part 2

The formulas that we use for decoding now have computer-checked proofs of working correctly for all inputs.

Interesting spinoff, unusual advantage of this cryptosystem: this decoding algorithm is rigid even without reencryption.

Some C functions for finite-field arithmetic and root-finding are computer-verified to match shorter descriptions in Cryptol for all inputs.

Software correctness, part 2

The formulas that we use for decoding now have computer-checked proofs of working correctly for all inputs.

Interesting spinoff, unusual advantage of this cryptosystem: this decoding algorithm is rigid even without reencryption.

Some C functions for finite-field arithmetic and root-finding are computer-verified to match shorter descriptions in Cryptol for all inputs.

Compiled software for the sorting subroutine has computer verification of sorting all inputs correctly.

Side note: scope of the Classic McEliece spec

ISO's "Performance principle": "Whenever possible, requirements shall be expressed in terms of performance rather than design or descriptive characteristics. This principle allows maximum freedom for technical development and reduces the risk of undesirable market impacts (e.g. limiting development of innovative solutions)."

Side note: scope of the Classic McEliece spec

ISO's "Performance principle": "Whenever possible, requirements shall be expressed in terms of performance rather than design or descriptive characteristics. This principle allows maximum freedom for technical development and reduces the risk of undesirable market impacts (e.g. limiting development of innovative solutions)."

IEEE "Guide for Developing System Requirements Specifications": "Each requirement should be implementation independent." Avoid the pitfall of including "implementation decisions along with the requirements statements".

Side note: scope of the Classic McEliece spec

ISO's "Performance principle": "Whenever possible, requirements shall be expressed in terms of performance rather than design or descriptive characteristics. This principle allows maximum freedom for technical development and reduces the risk of undesirable market impacts (e.g. limiting development of innovative solutions)."

IEEE "Guide for Developing System Requirements Specifications": "Each requirement should be implementation independent." Avoid the pitfall of including "implementation decisions along with the requirements statements".

The Classic McEliece specification says exactly what mathematical functions to compute. It does *not* constrain the algorithms used to compute those functions. There is a separate Classic McEliece guide for implementors.

Protection against timing attacks

The official Classic McEliece software has always avoided data-dependent branches and data-dependent array indices. The compiled software is now checked by TIMECOP 2, built into SUPERCOP.

Protection against timing attacks

The official Classic McEliece software has always avoided data-dependent branches and data-dependent array indices. The compiled software is now checked by TIMECOP 2, built into SUPERCOP.

Some big current issues regarding post-quantum implementation security:

- More instructions take variable time, as illustrated by KyberSlash exploiting divisions in Kyber reference code. Note that some CPUs even have variable-time multipliers.

Protection against timing attacks

The official Classic McEliece software has always avoided data-dependent branches and data-dependent array indices. The compiled software is now checked by TIMECOP 2, built into SUPERCOP.

Some big current issues regarding post-quantum implementation security:

- More instructions take variable time, as illustrated by KyberSlash exploiting divisions in Kyber reference code. Note that some CPUs even have variable-time multipliers.
- Compilers are introducing more and more timing variations, as illustrated by another attack demo against Kyber reference code.

Protection against timing attacks

The official Classic McEliece software has always avoided data-dependent branches and data-dependent array indices. The compiled software is now checked by TIMECOP 2, built into SUPERCOP.

Some big current issues regarding post-quantum implementation security:

- More instructions take variable time, as illustrated by KyberSlash exploiting divisions in Kyber reference code. Note that some CPUs even have variable-time multipliers.
- Compilers are introducing more and more timing variations, as illustrated by another attack demo against Kyber reference code.

McEliece software is naturally built from constant-time bit-vector operations. libmceliece has TIMECOP-like data-flow tests on the compiled library, and has proactive state-of-the-art source-level defenses (`cryptoint`).

Classic McEliece software for more and more environments

Ecosystem of unofficial libraries already available today:

- libmceliece: easy-to-use packaging for the official software.
- Debian and Ubuntu have integrated libmceliece.
- PQClean and liboqs have integrated the official software.
- libgcrypt has integrated the official software for `mceliece6688128`.
- pymceliece is a Python wrapper for libmceliece.
- node-mceliece-nist is a Node wrapper for the official software.
- classic-mceliece-rust is a Rust translation of the official software.
- Bouncy Castle includes Java and C# translations of the official software.
- McTiny supports tiny network servers.
- mceliece-arm-m4 supports ARM Cortex-M4 microcontrollers.
- McOutsourcing supports very-low-memory key generation.
- pqc-classic-mceliece implements Classic McEliece for FPGAs.

Examples of McEliece applications

Adva Network Security's high-speed optical networks:
"Our real-world use-case [for Classic McEliece] . . . Encrypted layer 1 optical transport solutions (OTNsec) with 10-400 Gbit/s including BSI approval".

Crypto4A's hardware security modules: "Crypto4A currently uses Classic McEliece in all of its HSMs for three important use cases".

Mullvad VPN software: 2022 announcement of Classic McEliece experiment on some servers, 2022 announcement of Classic McEliece experiment on all servers, 2023 announcement of stable support for Classic McEliece.

openssh-mceliece is a patch for OpenSSH to support `mceliece6688128`.

Rosenpass VPN software: uses `mceliece460896` for static keys.

Smoke secure-messaging system—although this has not upgraded yet from a different McEliece-based system to Classic McEliece.

Aren’t lattices more efficient?

One-time keys

NIST SP 800-57 Part 1, "Recommendation for Key Management: Part 1 – General": "Cryptoperiod: A public ephemeral key-agreement key is used for a single key-agreement transaction. The cryptoperiod of a public ephemeral key-agreement key ends immediately after it is used to generate a shared secret."

One-time keys

NIST SP 800-57 Part 1, "Recommendation for Key Management: Part 1 – General": "Cryptoperiod: A public ephemeral key-agreement key is used for a single key-agreement transaction. The cryptoperiod of a public ephemeral key-agreement key ends immediately after it is used to generate a shared secret."

Costs of using a KEM for a public one-time key-agreement key:
keygen cycles + pk bytes + enc cycles + ct bytes + dec cycles.

One-time keys

NIST SP 800-57 Part 1, “Recommendation for Key Management: Part 1 – General”: “Cryptoperiod: A public ephemeral key-agreement key is used for a single key-agreement transaction. The cryptoperiod of a public ephemeral key-agreement key ends immediately after it is used to generate a shared secret.”

Costs of using a KEM for a public one-time key-agreement key: keygen cycles + pk bytes + enc cycles + ct bytes + dec cycles.

In this scenario, `mceliece6960119` key agreement costs roughly 2^{-20} dollars (given current dollar costs of roughly 2^{-51} dollars per CPU cycle and roughly 2^{-40} dollars to send a byte through the Internet).

One-time keys

NIST SP 800-57 Part 1, "Recommendation for Key Management: Part 1 – General": "Cryptoperiod: A public ephemeral key-agreement key is used for a single key-agreement transaction. The cryptoperiod of a public ephemeral key-agreement key ends immediately after it is used to generate a shared secret."

Costs of using a KEM for a public one-time key-agreement key:
keygen cycles + pk bytes + enc cycles + ct bytes + dec cycles.

In this scenario, `mceliece6960119` key agreement costs roughly 2^{-20} dollars (given current dollar costs of roughly 2^{-51} dollars per CPU cycle and roughly 2^{-40} dollars to send a byte through the Internet).
A lattice key agreement costs only about 2^{-28} dollars.

Most types of public keys are not one-time keys

Non-ephemeral public-key types listed in NIST SP 800-57 Part 1:

- "Public signature-verification key": "The cryptoperiod may be on the order of several years".
- "Public authentication key": "An appropriate cryptoperiod for a public authentication key would be no more than one or two years".
- "Public key-transport key": "a recommendation for the cryptoperiod is no more than one or two years".
- "Public static key-agreement key": "The cryptoperiod of a public static key-agreement key may be one or two years".
- "Public authorization key": "no more than two years".

How should post-quantum cryptography handle these types of keys?

This question is not just about signatures

Signature-vs.-KEM choice for different types of static public keys:

- “Public signature-verification key”: Use signatures by definition.
  (Sometimes application needs offline signers or non-repudiation.)
- “Public key-transport key” or “public static key-agreement key”: Use KEMs.

This question is not just about signatures

Signature-vs.-KEM choice for different types of static public keys:

- "Public signature-verification key": Use signatures by definition. (Sometimes application needs offline signers or non-repudiation.)
- "Public key-transport key" or "public static key-agreement key": Use KEMs.
- "Public authentication key" or "public authorization key": Can use signatures, but can easily use KEMs instead. KEMs are typically more efficient (and give confidentiality as a bonus).

This question is not just about signatures

Signature-vs.-KEM choice for different types of static public keys:

- "Public signature-verification key": Use signatures by definition. (Sometimes application needs offline signers or non-repudiation.)
- "Public key-transport key" or "public static key-agreement key": Use KEMs.
- "Public authentication key" or "public authorization key": Can use signatures, but can easily use KEMs instead. KEMs are typically more efficient (and give confidentiality as a bonus).

STOC 1998 Bellare–Canetti–Krawczyk "A modular approach to the design and analysis of authentication and key exchange protocols": easily build public-key message authentication from public-key signatures *or* from public-key encryption.

KEM version: Bob encapsulates to Alice; Alice uses session key with a MAC.

This question is within scope

NIST's post-quantum CFP: "NIST intends to standardize post-quantum alternatives to its existing standards for digital signatures (FIPS 186) and key establishment (SP 800-56A, SP 800-56B). These standards are used in a wide variety of Internet protocols, such as TLS, SSH, IKE, IPsec, and DNSSEC. . . . The importance of public-key size may vary depending on the application; if applications can cache public keys, or otherwise avoid transmitting them frequently, the size of the public key may be of lesser importance."

This question is within scope

NIST's post-quantum CFP: "NIST intends to standardize post-quantum alternatives to its existing standards for digital signatures (FIPS 186) and key establishment (SP 800-56A, SP 800-56B). These standards are used in a wide variety of Internet protocols, such as TLS, SSH, IKE, IPsec, and DNSSEC. . . . The importance of public-key size may vary depending on the application; if applications can cache public keys, or otherwise avoid transmitting them frequently, the size of the public key may be of lesser importance."

NIST SP 800-56A says "can be a static key pair or an ephemeral key pair"; specifies static-static DH, static-ephemeral DH, ephemeral-ephemeral DH.

Note: Static-ephemeral DH can be viewed as KEM with static public key.

Many real-world examples of static public keys

Microsoft's Encrypting File System encrypts each file to the public key of the user authorized to access the file. This is more than just disk encryption: "Encrypting File System (EFS) can be used to encrypt files on a BitLocker-protected drive. BitLocker helps protect the entire operating system drive against offline attacks, whereas EFS can provide additional user-based file level encryption for security separation between multiple users of the same computer."

Many real-world examples of static public keys

Microsoft's Encrypting File System encrypts each file to the public key of the user authorized to access the file. This is more than just disk encryption: "Encrypting File System (EFS) can be used to encrypt files on a BitLocker-protected drive. BitLocker helps protect the entire operating system drive against offline attacks, whereas EFS can provide additional user-based file level encryption for security separation between multiple users of the same computer."

End-to-end email encryption (PGP etc.) is normally to a user's public key.

Many real-world examples of static public keys

Microsoft's Encrypting File System encrypts each file to the public key of the user authorized to access the file. This is more than just disk encryption: "Encrypting File System (EFS) can be used to encrypt files on a BitLocker-protected drive. BitLocker helps protect the entire operating system drive against offline attacks, whereas EFS can provide additional user-based file level encryption for security separation between multiple users of the same computer."

End-to-end email encryption (PGP etc.) is normally to a user's public key.

TLS server keys, authentication tokens, etc. typically use signatures, but would gain efficiency (and help move towards metadata confidentiality) by using post-quantum KEMs instead of post-quantum signatures.

Many real-world examples of static public keys

Microsoft's Encrypting File System encrypts each file to the public key of the user authorized to access the file. This is more than just disk encryption: "Encrypting File System (EFS) can be used to encrypt files on a BitLocker-protected drive. BitLocker helps protect the entire operating system drive against offline attacks, whereas EFS can provide additional user-based file level encryption for security separation between multiple users of the same computer."

End-to-end email encryption (PGP etc.) is normally to a user's public key.

TLS server keys, authentication tokens, etc. typically use signatures,
but would gain efficiency (and help move towards metadata confidentiality)
by using post-quantum KEMs instead of post-quantum signatures.

Note that TLS *CA keys* are different from TLS *server keys*.
The current TLS data flow needs TLS CA keys to be signature keys.

Impact of static keys on KEM efficiency analysis

Cost of client sending 2^{15} ciphertexts to server's static KEM key:
keygen cycles + pk bytes + $2^{15} \cdot$ (enc cycles + ct bytes + dec cycles).

Impact of static keys on KEM efficiency analysis

Cost of client sending 2^{15} ciphertexts to server's static KEM key:
keygen cycles + pk bytes + $2^{15} \cdot$ (enc cycles + ct bytes + dec cycles).

In this scenario, a lattice system costs roughly 2^{-14} dollars,
while total cost of `mceliece6960119` is only 2^{-17} dollars.
Reverses the cost winner compared to the situation of one-time keys.
Classic McEliece is the most efficient choice here, not just the safest!

Impact of static keys on KEM efficiency analysis

Cost of client sending 2^{15} ciphertexts to server's static KEM key:
keygen cycles + pk bytes + $2^{15} \cdot$ (enc cycles + ct bytes + dec cycles).

In this scenario, a lattice system costs roughly 2^{-14} dollars,
while total cost of `mceliece6960119` is only 2^{-17} dollars.
Reverses the cost winner compared to the situation of one-time keys.
Classic McEliece is the most efficient choice here, not just the safest!

As this illustrates, we can gain overall efficiency by paying attention to
the full range of key types already recognized in NIST SP 800-57,
not just static signature keys and one-time encryption keys.

Impact of static keys on KEM efficiency analysis

Cost of client sending 2^{15} ciphertexts to server's static KEM key:
keygen cycles + pk bytes + $2^{15} \cdot$ (enc cycles + ct bytes + dec cycles).

In this scenario, a lattice system costs roughly 2^{-14} dollars,
while total cost of `mceliece6960119` is only 2^{-17} dollars.
Reverses the cost winner compared to the situation of one-time keys.
Classic McEliece is the most efficient choice here, not just the safest!

As this illustrates, we can gain overall efficiency by paying attention to
the full range of key types already recognized in NIST SP 800-57,
not just static signature keys and one-time encryption keys.

(Same reversal appears when a popular server's public key is broadcast to
many different clients, even if key is changed every five minutes.)

Example: the Rosenpass VPN

> Rosenpass provides a complement to the well-known WireGuard protocol, adding quantum-hardened cryptography and key exchange while keeping the established WireGuard standard encryption security. So Rosenpass functions as an add-on, enhancing WireGuard's key negotiation process with Post Quantum Secure (PQS) cryptography, based a combination of Classic McEliece and Kyber.

Uses Classic McEliece for static keys, the foundation of security for identifying and authenticating the server, as well as for encrypting data. Ciphertexts are continually sent to those keys; the keys themselves are almost always cached.

Uses Kyber just for forward secrecy: a break of Kyber does not damage security unless the attacker can also steal secret keys through, e.g., hardware theft.

Trying to reuse Kyber for the static keys would increase security risks; increase costs; and consume extra key-exchange packets, making denial of service easier.

Summary of how to maximize efficiency

Most efficient post-quantum choice depends on the key type:

- "Public ephemeral key-agreement key":
  use KEMs; most efficient choice: lattices.
  (Pre-quantum uses ephemeral-ephemeral DH from SP 800-56A.)
- "Public key-transport key" or "public static key-agreement key":
  use KEMs; most efficient choice: **Classic McEliece**.
  (Pre-quantum uses static-ephemeral DH from SP 800-56A.)
- "Public authentication key" or "public authorization key":
  can use signatures or KEMs; most efficient choice: **Classic McEliece**.
  (Pre-quantum again uses static-ephemeral DH from SP 800-56A.)
- "Public signature-verification key":
  use signatures; most efficient choice: outside scope of this talk.

Recommendations

We recommend recognizing the importance of static encryption keys, as in NIST SP 800-56A, NIST SP 800-57, and many applications.

Independently of KEM choices made for ephemeral encryption keys, we recommend standardizing Classic McEliece as the best option for static encryption keys: the safest and most efficient option.

(We also recommend recognizing the intermediate case of keys that are periodically rotated for forward secrecy but still used many times.)