The McEliece cryptosystem

Daniel J. Bernstein

Can cryptographers be trusted?

48% of the 69 round-1 submissions to the NIST post-quantum competition in 2017 are broken by now. Some of the attacks finish in seconds.

Can cryptographers be trusted?

48% of the 69 round-1 submissions to the NIST post-quantum competition in 2017 are broken by now. Some of the attacks finish in seconds.

25% of the 48 submissions that were not broken during round 1 are broken by now.

Can cryptographers be trusted?

48% of the 69 round-1 submissions to the NIST post-quantum competition in 2017 are broken by now. Some of the attacks finish in seconds.

25% of the 48 submissions that were not broken during round 1 are broken by now.

36% of the 28 submissions selected by NIST in 2019 for round 2 are broken by now.

Can cryptographers be trusted?

48% of the 69 round-1 submissions to the NIST post-quantum competition in 2017 are broken by now. Some of the attacks finish in seconds.

25% of the 48 submissions that were not broken during round 1 are broken by now.

36% of the 28 submissions selected by NIST in 2019 for round 2 are broken by now.

See `https://cr.yp.to/papers.html#qrcsp` for data, references, statistical-significance analysis.

Are cryptographers prioritizing security?

1991: NIST proposed a signature standard without mentioning that it had been secretly designed by NSA. The proposal had many weaknesses, including a 2^{64} attack that the public didn't find until 2000.

Are cryptographers prioritizing security?

1991: NIST proposed a signature standard without mentioning that it had been secretly designed by NSA. The proposal had many weaknesses, including a 2^{64} attack that the public didn't find until 2000.

2013 news report: NSA paid the RSA company $10 million to make NSA's backdoored Dual EC RNG the default RNG in RSA's BSafe software.

Are cryptographers prioritizing security?

1991: NIST proposed a signature standard without mentioning that it had been secretly designed by NSA. The proposal had many weaknesses, including a 2^{64} attack that the public didn't find until 2000.

2013 news report: NSA paid the RSA company $10 million to make NSA's backdoored Dual EC RNG the default RNG in RSA's BSafe software.

2019: Aumasson recommends using ChaCha8 instead of ChaCha20 "for a future where less energy is wasted on computing superfluous rounds".

Performance vs. security

Computer-science courses highlight performance.
Prioritizing performance often reduces security. Bad!

Performance vs. security

Computer-science courses highlight performance.
Prioritizing performance often reduces security. Bad!

But improving performance *while maintaining high security* can enable security deployment. Good!

Performance vs. security

Computer-science courses highlight performance. Prioritizing performance often reduces security. Bad!

But improving performance *while maintaining high security* can enable security deployment. Good!

Beware overconfidence. Example: For ChaCha8, Aumasson says “we have strong confidence that the algorithm will never be wounded”.

Performance vs. security

Computer-science courses highlight performance.
Prioritizing performance often reduces security. Bad!

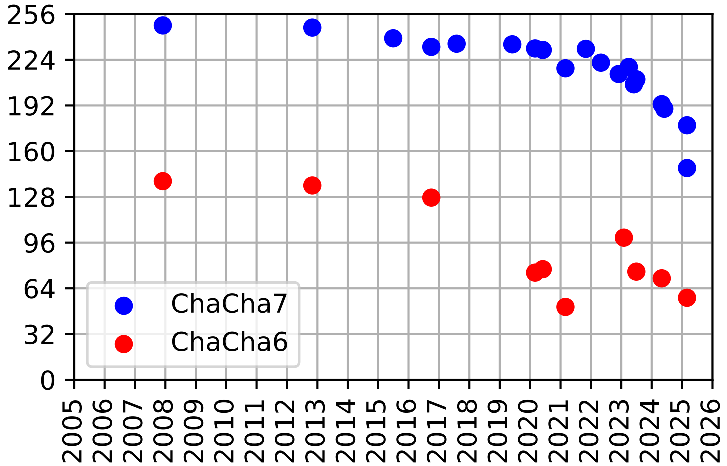
But improving performance *while maintaining high security* can enable security deployment. Good!

Beware overconfidence. Example: For ChaCha8, Aumasson says "we have strong confidence that the algorithm will never be wounded".

I say: Use ChaCha20 *except* when deployment really needs the speed of ChaCha12 or ChaCha8.

Attack costs for reduced-round ChaCha

Horizontal axis: date of publication. Vertical axis: reported attack cost. ChaCha has 256-bit keys.



Be careful with risk assessment

For Aumasson, “wounded” is *much* faster than 2^{256}.
ChaCha6 has been “wounded”; ChaCha7 hasn’t.

Each added round makes attacks more challenging.
Maybe ChaCha8 will never be “wounded”.
But it doesn’t have a comfortable security margin.

Be careful with risk assessment

For Aumasson, “wounded” is *much* faster than 2^{256}.
ChaCha6 has been “wounded”; ChaCha7 hasn’t.

Each added round makes attacks more challenging.
Maybe ChaCha8 will never be “wounded”.
But it doesn’t have a comfortable security margin.

The 2019 claim of “strong confidence” comes from

- looking at several attack papers over 10 years that find only small improvements, and
- assuming that 10 years is long enough.

Why McEliece?

Reason #1 for McEliece: security

Out of all public-key encryption systems,
the McEliece system **minimizes security risks**.
We'll see various ways to quantify this.

Reason #2 for McEliece: performance

The McEliece system has very small ciphertexts:
e.g., 96 bytes for `mceliece348864`.
We'll see why NTRU, Kyber, etc. are bigger.

(ECC is smaller, but broken by Shor's algorithm.)

Reason #2 for McEliece: performance

The McEliece system has very small ciphertexts:
e.g., 96 bytes for `mceliece348864`.
We'll see why NTRU, Kyber, etc. are bigger.

(ECC is smaller, but broken by Shor's algorithm.)

McEliece enc is simple, small (streaming public keys), and fast: e.g., on Skylake, 30k cycles for `mceliece348864` vs. 36k for `kyber512`.

McEliece dec is more complicated but still fast:
e.g., 118k Skylake cycles for `mceliece348864`.

“No, McEliece performance is bad!”

`mceliece348864` has 261120-byte keys.
Key generation is 31 Mcycles on Skylake.

“No, McEliece performance is bad!”

`mceliece348864` has 261120-byte keys.
Key generation is 31 Mcycles on Skylake.

(A 2025 NIST report, hiding a keygen speedup published by Tung Chou in 2019, says 114 Mcycles. The page cited by NIST says that it comes from a “defunct” benchmarking project and that its “measurements are not up to date”. The page already had these warnings in April 2024. Years of official submissions to NIST had the new numbers.)

“No, McEliece performance is bad!”

`mceliece348864` has 261120-byte keys.
Key generation is 31 Mcycles on Skylake.

(A 2025 NIST report, hiding a keygen speedup published by Tung Chou in 2019, says 114 Mcycles. The page cited by NIST says that it comes from a “defunct” benchmarking project and that its “measurements are not up to date”. The page already had these warnings in April 2024. Years of official submissions to NIST had the new numbers.)

Recommended key sizes for long-term security are 1MB, with 197 Mcycles for keygen on Skylake.

Understanding how applications use keys

Cryptographic applications use different types of keys, typically combining multiple cryptosystems.

Example: applications using RSA encryption normally use RSA to encrypt a random session key, and then use an authenticated cipher such as ChaCha20-Poly1305 to encrypt the user's message.

Understanding how applications use keys

Cryptographic applications use different types of keys, typically combining multiple cryptosystems.

Example: applications using RSA encryption normally use RSA to encrypt a random session key, and then use an authenticated cipher such as ChaCha20-Poly1305 to encrypt the user's message.

— "Wouldn't it be simpler to skip the cipher? Just use RSA to encrypt each block of the message."

Understanding how applications use keys

Cryptographic applications use different types of keys, typically combining multiple cryptosystems.

Example: applications using RSA encryption normally use RSA to encrypt a random session key, and then use an authenticated cipher such as ChaCha20-Poly1305 to encrypt the user's message.

— "Wouldn't it be simpler to skip the cipher? Just use RSA to encrypt each block of the message."

— This would be slower *and* more likely to have security problems. RSA is hard to get right; focusing on random inputs removes some of the issues.

Multiple types of public keys

The simplest use of public-key encryption:
All clients know the server's "static encryption key".

Multiple types of public keys

The simplest use of public-key encryption:
All clients know the server's "static encryption key".

More complicated public-key infrastructure:
All clients know an authority's "static signature key". Authority signs server's static encryption key.

Multiple types of public keys

The simplest use of public-key encryption:
All clients know the server's "static encryption key".

More complicated public-key infrastructure:
All clients know an authority's "static signature key". Authority signs server's static encryption key.

More complicated protection against server theft:
The server creates an "ephemeral encryption key" and erases it after 10 minutes or after 100 uses.
(Most extreme case: "one-time encryption key".)
Also use static keys to protect the ephemeral keys!

Counting bytes in multiple scenarios

	`kyber512`	`mceliece348864`
Ciphertext	768	96
Public key	800	261120
Server sends 1000 **one-time** public keys, client sends 1000 ciphertexts	1568000	261216000
♻ Server sends a **static** public key, client sends 1000 ciphertexts	768800	357120

Choosing post-quantum cryptosystems

Most applications aren't bottlenecks for most users.
So prioritize security! Use McEliece.

Choosing post-quantum cryptosystems

Most applications aren't bottlenecks for most users. So prioritize security! Use McEliece.

Even when an application is a bottleneck for the user, the bottleneck in that application is usually not post-quantum cryptography. Use McEliece.

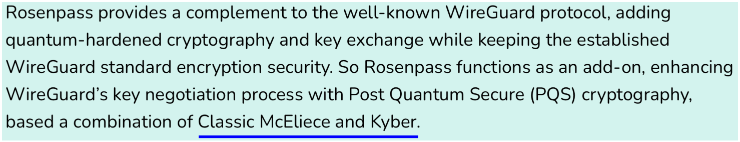
Choosing post-quantum cryptosystems

Most applications aren't bottlenecks for most users. So prioritize security! Use McEliece.

Even when an application is a bottleneck for the user, the bottleneck in that application is usually not post-quantum cryptography. Use McEliece.

For an application where the cost of post-quantum cryptography really is an issue, McEliece is generally the most efficient option for static keys, but ephemeral keys are forced to use something riskier.

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 Kyber only for forward secrecy: ephemeral keys to protect past communication if server is stolen.

Uses McEliece for static keys to identify the server, to authenticate the server, and to encrypt data. Ciphertexts are continually sent to those keys; the keys themselves are almost always cached. Using Kyber for these keys would add cost (+ risk).

More examples of McEliece deployment

Adva Network Security's high-speed optical networks: "Our real-world use-case . . . 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: 2022 announcement of McEliece experiment on some servers, 2022 announcement of McEliece experiment on all servers, 2023 announcement of stable support for McEliece.

Implementations for many environments

Easy-to-use software library libmceliece has already been integrated into Debian and Ubuntu.
pymceliece is a Python wrapper. node-mceliece-nist is a Node wrapper. classic-mceliece-rust is a Rust translation. Botan includes a C++ translation.
Bouncy Castle includes Java and C# translations.
libgcrypt includes `mceliece6688128`. McTiny supports tiny network servers. mceliece-arm-m4 supports ARM Cortex-M4 microcontrollers.
McOutsourcing supports very-low-memory key generation. pqc-classic-mceliece supports FPGAs.
See `https://mceliece.org` for more resources.

How standards are selected

NIST asks for McEliece use cases

NIST, 2022: "NIST would like feedback on specific use cases for which Classic McEliece would be a good solution."

Immediate responses and subsequent responses:

- examples where McEliece is already being used;
- more examples where McEliece would be good;
- reasons for the interest in McEliece.

An example of the responses

Official comment from telecom company Ericsson, 2024: “We strongly think NIST should standardize Classic McEliece, which has properties that makes it the best choice in many different applications. We are planning to use Classic McEliece. . . . The small ciphertexts and good performance makes Classic McEliece the best choice for many applications of static encapsulation keys of which there are many (WireGuard, S/MIME, IMSI encryption, File encryption, Noise, EDHOC, etc.). . . .”

“Oops, we weren’t expecting answers!”

“Oops, we weren’t expecting answers!”

NIST’s 2025 report says: “NIST requested feedback on specific use cases for which Classic McEliece would be a good solution.”

Of course, the report then cites the responses, and says that, yes, there is already McEliece usage plus strong interest in McEliece standardization.

“Oops, we weren’t expecting answers!”

NIST’s 2025 report says: “NIST requested feedback on specific use cases for which Classic McEliece would be a good solution.”

~~Of course, the report then cites the responses, and says that, yes, there is already McEliece usage plus strong interest in McEliece standardization.~~

Actually, the report doesn’t do that.

"Oops, we weren't expecting answers!"

NIST's 2025 report says: "NIST requested feedback on specific use cases for which Classic McEliece would be a good solution."

~~Of course, the report then cites the responses, and says that, yes, there is already McEliece usage plus strong interest in McEliece standardization.~~

Actually, the report doesn't do that. It says: "Responses noted that Classic McEliece **may** provide better performance **than BIKE or HQC** for [static keys] . . . the interest expressed in Classic McEliece was limited". (Emphasis added.)

Interesting new terminology in the report

Old name	New name
One-time keys	“general applications”
Static keys	“applications in which a public key can be transferred once and then used for several encapsulations”

(Exception: the old terminology is used in one line on page 12; probably an editing error.)

Interesting new terminology in the report

Old name	New name
One-time keys	“general applications”
Static keys	“applications in which a public key can be transferred once and then used for several encapsulations”

(Exception: the old terminology is used in one line on page 12; probably an editing error.)

Saying “the most efficient option for static keys is McEliece” would make it difficult to use efficiency as an excuse to delay McEliece standardization. In the report, McEliece’s efficiency advantage sounds obscure and uncertain.

Interesting approach to consistency

Same report: “Having more standards to implement adds complexity to protocols and PQC migration.”

Interesting approach to consistency

Same report: “Having more standards to implement adds complexity to protocols and PQC migration.”

For post-quantum signatures, NIST has already standardized Dilithium, LMS, SPHINCS+, and XMSS; says it will standardize Falcon “because its small bandwidth may be necessary in certain applications”; and is asking for more options, including small-signature large-key options.

Interesting approach to consistency

Same report: "Having more standards to implement adds complexity to protocols and PQC migration."

For post-quantum signatures, NIST has already standardized Dilithium, LMS, SPHINCS+, and XMSS; says it will standardize Falcon "because its small bandwidth may be necessary in certain applications"; and is asking for more options, including small-signature large-key options.

NIST's official competition rules said "The goal of this process is to select a number of acceptable candidate cryptosystems for standardization".

Interesting approach to evidence

The report says that HQC's "performance profile would be acceptable for most general applications", making HQC a "general-purpose" KEM.

In 2020, NIST portrayed FrodoKEM as not having "acceptable performance in widely used applications overall", so NIST eliminated FrodoKEM.

Interesting approach to evidence

The report says that HQC's "performance profile would be acceptable for most general applications", making HQC a "general-purpose" KEM.

In 2020, NIST portrayed FrodoKEM as not having "acceptable performance in widely used applications overall", so NIST eliminated FrodoKEM.

So HQC-1 (2249-byte pk, 4497-byte ct) has acceptable performance for most applications, while FrodoKEM-640 (9616-byte pk, 9720-byte ct) has unacceptable performance for most applications? Where is NIST's list of per-application budgets?

NIST's decisions are controversial

ISO is currently considering a draft standard that includes McEliece, FrodoKEM, and Kyber.

NIST's decisions are controversial

ISO is currently considering a draft standard that includes McEliece, FrodoKEM, and Kyber.

NIST's report says "After the ISO standardization process has been completed, NIST may consider developing a standard for Classic McEliece based on the ISO standard".

NIST's decisions are controversial

ISO is currently considering a draft standard that includes McEliece, FrodoKEM, and Kyber.

NIST's report says "After the ISO standardization process has been completed, NIST may consider developing a standard for Classic McEliece based on the ISO standard".

But NIST could already have standardized McEliece years ago. For some reason, NIST is trying very hard to push everyone into using Kyber.

Understanding who's in charge

A 2014 NIST presentation listed its authorship as "Post Quantum Cryptography Team, National Institute of Standards and Technology (NIST), pqc@nist.gov". See also NIST's 2017 annual report.

Understanding who's in charge

A 2014 NIST presentation listed its authorship as "Post Quantum Cryptography Team, National Institute of Standards and Technology (NIST), pqc@nist.gov". See also NIST's 2017 annual report.

A lawsuit has revealed the list of people who were members of pqc@nist.gov in 2016. There were more NSA members (Scott Simon, Nick Gajcowski, Mark Motley, Laurie Law, John McVey, Jerry Solinas, Daniel Kirkwood, David Tuller, David Hubbard, Bradley C. Lackey) than NIST members.

What's inside McEliece

One way to invent the McEliece system

0. Start with modern lattice-based cryptography.

One way to invent the McEliece system

0. Start with modern lattice-based cryptography.

1. Choose **modulus 2**. Bad: slower in software. Good: simpler; easier analysis; much more stability against cryptanalysis; nicer for hardware.

One way to invent the McEliece system

0. Start with modern lattice-based cryptography.

1. Choose **modulus 2**. Bad: slower in software.
Good: simpler; easier analysis; much more stability against cryptanalysis; nicer for hardware.

2. Then switch to a **more powerful decoder**.
Bad: more complicated decoding algorithm.
Good: much better security vs. ciphertext size.

One way to invent the McEliece system

0. Start with modern lattice-based cryptography.

1. Choose **modulus 2**. Bad: slower in software.
Good: simpler; easier analysis; much more stability against cryptanalysis; nicer for hardware.

2. Then switch to a **more powerful decoder**.
Bad: more complicated decoding algorithm.
Good: much better security vs. ciphertext size.

3. Then **travel back in time** to publish in 1978.
Good: allows half century of security analysis and half century of implementation improvements.

The general framework

System parameters: $n, q, r \in \{1, 2, 3, \ldots\}$.

Public key determines $K_0, \ldots, K_{n-1} \in (\mathbb{Z}/q)^r$.
Notation: $\mathbb{Z}/q$ is the ring of integers mod q;
$(\mathbb{Z}/q)^r = \{(u_0, \ldots, u_{r-1}) : \text{each } u_i \in \mathbb{Z}/q\}$;
$a, b \in X$ means $a \in X$ and $b \in X$.

The general framework

System parameters: $n, q, r \in \{1, 2, 3, \ldots\}$.

Public key determines $K_0, \ldots, K_{n-1} \in (\mathbb{Z}/q)^r$.
Notation: $\mathbb{Z}/q$ is the ring of integers mod q;
$(\mathbb{Z}/q)^r = \{(u_0, \ldots, u_{r-1}) : \text{each } u_i \in \mathbb{Z}/q\}$;
$a, b \in X$ means $a \in X$ and $b \in X$.

Ciphertext: $C = s_0K_0 + \cdots + s_{n-1}K_{n-1} \in (\mathbb{Z}/q)^r$
where $s_0, \ldots, s_{n-1} \in \mathbb{Z}$ are small secrets.
Ciphertext has $r \log_2 q$ bits.

The general framework

System parameters: $n, q, r \in \{1, 2, 3, \ldots\}$.

Public key determines $K_0, \ldots, K_{n-1} \in (\mathbb{Z}/q)^r$.
Notation: $\mathbb{Z}/q$ is the ring of integers mod q;
$(\mathbb{Z}/q)^r = \{(u_0, \ldots, u_{r-1}) : \text{each } u_i \in \mathbb{Z}/q\}$;
$a, b \in X$ means $a \in X$ and $b \in X$.

Ciphertext: $C = s_0K_0 + \cdots + s_{n-1}K_{n-1} \in (\mathbb{Z}/q)^r$
where $s_0, \ldots, s_{n-1} \in \mathbb{Z}$ are small secrets.
Ciphertext has $r \log_2 q$ bits.

This covers "code-based" and "lattice-based" encryption. Let's call this **cola encryption**.

A cola example: `ntruhps2048509`

System parameters: $(n, q, r) = (1018, 2048, 508)$.

Public key determines $K_0, \ldots, K_{1017} \in (\mathbb{Z}/2048)^{508}$.

Ciphertext: $C = s_0K_0 + \cdots + s_{1017}K_{1017}$
for secrets $s_0, \ldots, s_{1017} \in \{-1, 0, 1\}$.

Ciphertext has $508 \log_2 2048 = 5588$ bits,
i.e., $5588/8 = 698.5$ bytes, sent in 699 bytes.

(Exercise: What are n, q, r for `kyber512`?)

Lattice attacks

Attacker sees $C, K_0, \ldots, K_{n-1} \in (\mathbb{Z}/q)^r$.
Easy linear-algebra computation finds big
$t_0, \ldots, t_{n-1} \in \mathbb{Z}$ with $C = t_0 K_0 + \cdots + t_{n-1} K_{n-1}$.

Lattice attacks

Attacker sees $C, K_0, \ldots, K_{n-1} \in (\mathbb{Z}/q)^r$.
Easy linear-algebra computation finds big $t_0, \ldots, t_{n-1} \in \mathbb{Z}$ with $C = t_0 K_0 + \cdots + t_{n-1} K_{n-1}$.

Note: $(t_0 - s_0)K_0 + \cdots + (t_{n-1} - s_{n-1})K_{n-1} = 0$;
i.e., $(t_0 - s_0, \ldots, t_{n-1} - s_{n-1}) \in L$ where $L = \{(v_0, \ldots, v_{n-1}) \in \mathbb{Z}^n : v_0 K_0 + \cdots + v_{n-1} K_{n-1} = 0\}$.

Lattice attacks

Attacker sees $C, K_0, \dots, K_{n-1} \in (\mathbb{Z}/q)^r$.
Easy linear-algebra computation finds big
$t_0, \dots, t_{n-1} \in \mathbb{Z}$ with $C = t_0K_0 + \dots + t_{n-1}K_{n-1}$.

Note: $(t_0 - s_0)K_0 + \dots + (t_{n-1} - s_{n-1})K_{n-1} = 0$;
i.e., $(t_0 - s_0, \dots, t_{n-1} - s_{n-1}) \in L$ where $L =$
$\{(v_0, \dots, v_{n-1}) \in \mathbb{Z}^n : v_0K_0 + \dots + v_{n-1}K_{n-1} = 0\}$.

Attack problem is now a "close-vector problem":
find v in lattice L with $v \approx (t_0, \dots, t_{n-1})$.

NP-hardness myths for lattice encryption

Standard conjectures: "the polynomial hierarchy does not collapse"; in particular, P $\neq$ NP; so, for every NP-hard problem, every poly-time algorithm fails to solve **some** example of the problem.

NP-hardness myths for lattice encryption

Standard conjectures: “the polynomial hierarchy does not collapse”; in particular, P $\neq$ NP; so, for every NP-hard problem, every poly-time algorithm fails to solve **some** example of the problem.

Fact: The general problem of finding $v \in L$ with $v \approx t$ is NP-hard. (1981 van Emde Boas)

NP-hardness myths for lattice encryption

Standard conjectures: “the polynomial hierarchy does not collapse”; in particular, $P \neq NP$; so, for every NP-hard problem, every poly-time algorithm fails to solve **some** example of the problem.

Fact: The general problem of finding $v \in L$ with $v \approx t$ is NP-hard. (1981 van Emde Boas)

Common mistake: “Attacking lattice encryption is an example of this problem, so it’s NP-hard.”

No, there’s no reason to think attacking lattice encryption is NP-hard. Fact: Every problem broken in poly time is an example of an NP-hard problem.

Lattice gaps

Picture from 2005 Aharanov–Regev:

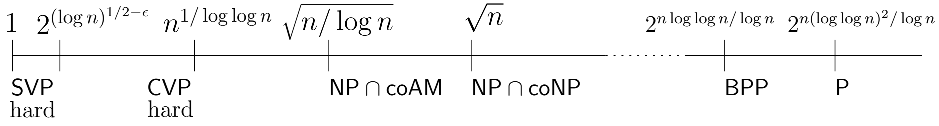


Figure 1: The complexity of lattice problems (some constants omitted)

Lattice gaps

Picture from 2005 Aharanov–Regev:

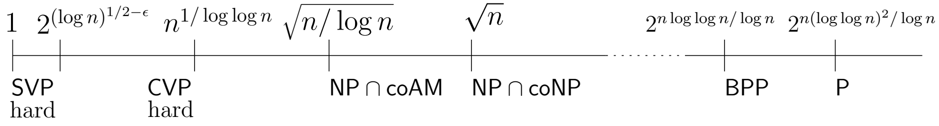


Figure 1: The complexity of lattice problems (some constants omitted)

Right side, large "gap": t is particularly close to L; fast algorithms find closest vector.

Lattice gaps

Picture from 2005 Aharanov–Regev:

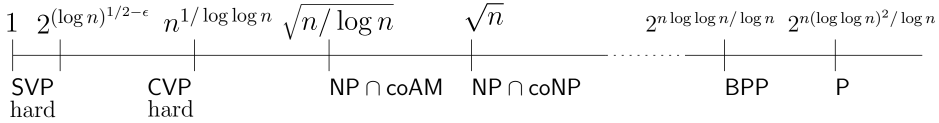


Figure 1: The complexity of lattice problems (some constants omitted)

Right side, large "gap": t is particularly close to L; fast algorithms find closest vector.

Left side, small "gap": t is far from L; NP-hard.

Lattice gaps

Picture from 2005 Aharanov–Regev:

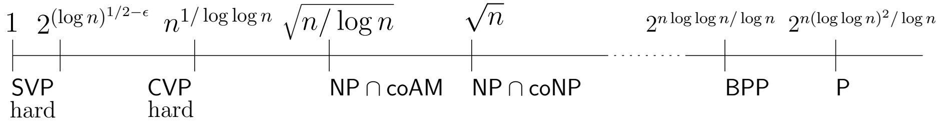


Figure 1: The complexity of lattice problems (some constants omitted)

Right side, large "gap": t is particularly close to L; fast algorithms find closest vector.

Left side, small "gap": t is far from L; NP-hard.

Middle: the standard conjectures imply that the problem is *not* NP-hard for, e.g., "gap" $\sqrt{n}$.

Warning: multiple gap concepts

A typical "decisional" gap problem:
you're guaranteed that *either* $\text{dist}(t, L) \le d/G$ *or* $\text{dist}(t, L) > d$; problem is to figure out which.

Warning: multiple gap concepts

A typical “decisional” gap problem:
you’re guaranteed that *either* $\text{dist}(t, L) \le d/G$ *or* $\text{dist}(t, L) > d$; problem is to figure out which.

“Search” problem: find $v \in L$ with $\text{dist}(t, v) \le d/G$ given that all other $w \in L$ have $\text{dist}(t, w) > d$.

Warning: multiple gap concepts

A typical "decisional" gap problem:
you're guaranteed that *either* $\operatorname{dist}(t, L) \le d/G$ *or* $\operatorname{dist}(t, L) > d$; problem is to figure out which.

"Search" problem: find $v \in L$ with $\operatorname{dist}(t, v) \le d/G$ given that all other $w \in L$ have $\operatorname{dist}(t, w) > d$.

Suffices to solve an "approximation" problem:
find $v \in L$ with $\operatorname{dist}(t, v) \le G \operatorname{dist}(t, L)$.

Warning: multiple gap concepts

A typical "decisional" gap problem:
you're guaranteed that *either* $\text{dist}(t, L) \le d/G$ *or* $\text{dist}(t, L) > d$; problem is to figure out which.

"Search" problem: find $v \in L$ with $\text{dist}(t, v) \le d/G$ given that all other $w \in L$ have $\text{dist}(t, w) > d$.

Suffices to solve an "approximation" problem:
find $v \in L$ with $\text{dist}(t, v) \le G\,\text{dist}(t, L)$.

If $d = \max\{\text{dist}(u, L)\}$ then the guarantee forces $\text{dist}(t, L) \le d/G$ so $G \le \max\{\text{dist}(u, L)\}/\text{dist}(t, L)$.
For simplicity, let's focus on computing this cutoff gap: $\max\{\text{dist}(u, L)\}/\text{dist}(t, L)$.

What's the NTRU cutoff gap?

NTRU has $s_0, \ldots, s_{n-1} \in \{-1, 0, 1\}$,
so $\operatorname{dist}(t, L) \le |(s_0, \ldots, s_{n-1})| \le n^{1/2}$.

What's the NTRU cutoff gap?

NTRU has $s_0, \ldots, s_{n-1} \in \{-1, 0, 1\}$,
so $\operatorname{dist}(t, L) \le |(s_0, \ldots, s_{n-1})| \le n^{1/2}$.

Typically q is chosen as $\Theta(n)$. Can then show that most vectors have distance $\Omega(n)$ from L, so cutoff gap is $\Omega(n)/n^{1/2}$, i.e., $\Omega(n^{1/2})$.

(Exercise: Prove this gap.)

What's the NTRU cutoff gap?

NTRU has $s_0, \ldots, s_{n-1} \in \{-1, 0, 1\}$,
so $\operatorname{dist}(t, L) \le |(s_0, \ldots, s_{n-1})| \le n^{1/2}$.

Typically q is chosen as $\Theta(n)$. Can then show that most vectors have distance $\Omega(n)$ from L, so cutoff gap is $\Omega(n)/n^{1/2}$, i.e., $\Omega(n^{1/2})$.

(Exercise: Prove this gap.)

This doesn't mean NTRU is broken! Maybe attacking NTRU is hard without being NP-hard.

The NTRU decoder

Alice generates an NTRU secret key and a public key determining $K_0, \ldots, K_{n-1} \in (\mathbb{Z}/q)^r$.

The secret key determines a linear transformation φ such that $\varphi(K_0), \ldots, \varphi(K_{n-1})$ are small.

The NTRU decoder

Alice generates an NTRU secret key and a public key determining $K_0, \ldots, K_{n-1} \in (\mathbb{Z}/q)^r$.

The secret key determines a linear transformation φ such that $\varphi(K_0), \ldots, \varphi(K_{n-1})$ are small.

Bob computes $C = s_0 K_0 + \cdots + s_{n-1} K_{n-1}$. Alice computes $\varphi(C) = s_0 \varphi(K_0) + \cdots + s_{n-1} \varphi(K_{n-1})$, which is small, so the reduction mod q disappears. A fast algorithm solves for $s_0, \ldots, s_{n-1}$.

A cola example mod 2: `bikel1`

System parameters: $(n, q, r) = (24646, 2, 12323)$.

Public key determines $K_0, \ldots, K_{24645} \in (\mathbb{Z}/2)^{12323}$.

Ciphertext: $C = s_0 K_0 + \cdots + s_{24645} K_{24645}$
for "weight-134" vector $(s_0, \ldots, s_{24645}) \in \{0, 1\}$;
i.e., $\#\{i : s_i \neq 0\} = 134$.
Ciphertext has 12323 bits $\approx$ 1541 bytes.

Alice generated weight-71 $\varphi(K_0), \ldots, \varphi(K_{24645})$.
Then $\varphi(C) = s_0\varphi(K_0) + \cdots + s_{24645}\varphi(K_{24645})$
involves some reductions mod 2, but
fast statistics usually solve for $s_0, \ldots, s_{24645}$.

What's the BIKE cutoff gap?

BIKE takes $(s_0, \ldots, s_{n-1})$ of weight $\Theta(n^{1/2})$, so t has distance $\Theta(n^{1/4})$ from lattice L.

Can show that most vectors have distance $\Theta(n^{1/2})$ from L, so cutoff gap is $\Theta(n^{1/4})$.

What's the BIKE cutoff gap?

BIKE takes $(s_0, \ldots, s_{n-1})$ of weight $\Theta(n^{1/2})$, so t has distance $\Theta(n^{1/4})$ from lattice L.

Can show that most vectors have distance $\Theta(n^{1/2})$ from L, so cutoff gap is $\Theta(n^{1/4})$.

Compared to NTRU:

- Gap sounds smaller. More secure?
- But t sounds closer to L. Fewer s possibilities. Less secure?

`ntruhps2048509` (699-byte ciphertexts) and `bikel1` (1541-byte ciphertexts) are both designed to have roughly 128 bits of security.

The basic ISD attack

There are $\binom{n}{w}$ weight-w vectors $s \in (\mathbb{Z}/2)^n$.

For $(n, w) = (24646, 134)$: $\binom{n}{w} \approx 2^{1196}$.

The basic ISD attack

There are $\binom{n}{w}$ weight-w vectors $s \in (\mathbb{Z}/2)^n$.
For $(n, w) = (24646, 134)$: $\binom{n}{w} \approx 2^{1196}$.

Faster than searching through all s:
1962 Prange "information-set decoding".
Basic idea: Maybe $s_r = s_{r+1} = \cdots = s_{n-1} = 0$;
probability $\binom{r}{w}/\binom{n}{w} \approx 2^{-134.52}$.
Then $C = s_0K_0 + \cdots + s_{r-1}K_{r-1}$.
Solve for $s_0, \ldots, s_{r-1}$ by linear algebra.
If this fails, permute $\{0, \ldots, n-1\}$ and try again.

The basic ISD attack

There are $\binom{n}{w}$ weight-w vectors $s \in (\mathbb{Z}/2)^n$.
For $(n, w) = (24646, 134)$: $\binom{n}{w} \approx 2^{1196}$.

Faster than searching through all s:
1962 Prange "information-set decoding".
Basic idea: Maybe $s_r = s_{r+1} = \cdots = s_{n-1} = 0$;
probability $\binom{r}{w}/\binom{n}{w} \approx 2^{-134.52}$.
Then $C = s_0 K_0 + \cdots + s_{r-1} K_{r-1}$.
Solve for $s_0, \ldots, s_{r-1}$ by linear algebra.
If this fails, permute $\{0, \ldots, n-1\}$ and try again.

See https://isd.mceliece.org for 50 papers studying ISD. Noticeable speedups, mostly in linear algebra. No change in asymptotic attack exponent.

NTRU security vs. BIKE security

NTRU has 3^n possible choices of s encrypted as $r \log_2 q \approx (n/2) \log_2 n$ ciphertext bits.

e.g. `ntruhps2048509`: $3^{1018} \approx 2^{1613}$ choices of s encrypted as 5588 ciphertext bits.

Compared to BIKE, less information about more choices of s. Why isn't this a higher security level?

NTRU security vs. BIKE security

NTRU has 3^n possible choices of s encrypted as $r \log_2 q \approx (n/2) \log_2 n$ ciphertext bits.

e.g. `ntruhps2048509`: $3^{1018} \approx 2^{1613}$ choices of s encrypted as 5588 ciphertext bits.

Compared to BIKE, less information about more choices of s. Why isn't this a higher security level?

Answer: NTRU attacks use combinatorial searches and linear algebra *and* size variations mod q.
Size variations have led to big attack speedups.

Another cola example: `mceliece348864`

System parameters: $(n, q, r) = (3488, 2, 768)$.

Public key determines $K_0, \ldots, K_{3487} \in (\mathbb{Z}/2)^{768}$.

Ciphertext: $C = s_0 K_0 + \cdots + s_{3487} K_{3487}$
for weight-64 vector $(s_0, \ldots, s_{3487}) \in \{0, 1\}$.
Ciphertext has 768 bits, i.e., 96 bytes.

This encrypts $\binom{3488}{64} \approx 2^{456}$ choices of s into just 768 bits. Alice decrypts using a more powerful decoder than the NTRU or BIKE decoders.

This is another system designed for 128-bit security.
Prange uses $\binom{n}{64}/\binom{r}{64} \approx 2^{142.78}$ iterations.

What's the McEliece cutoff gap?

Normally take $n \approx 5r$, weight $w \approx 0.2n/\log_2 n$.

Now $|s| = w^{1/2} \in \Theta(n^{1/2}/(\log n)^{1/2})$.

Can show that most vectors have distance $\Theta(n^{1/2})$ from L. Gap is just $\Theta((\log n)^{1/2})$.
"Polylog-gap poly-distance cola encryption".

i.e.: t is *almost* as far from L as most vectors are.
This relies critically on the power of Alice's decoder!

Summary of numerical features

Comparing PKEs (public-key encryption systems) by orders of magnitude of $|s|$ etc.:

PKE	q	ct size	$\lvert s\rvert$	cutoff gap
NTRU	n	$n \log n$	$n^{1/2}$	$n^{1/2}$
BIKE	2	n	$n^{1/4}$	$n^{1/4}$
McEliece	2	n	$(n/\log n)^{1/2}$	$(\log n)^{1/2}$

Summary of numerical features

Comparing PKEs (public-key encryption systems) by orders of magnitude of $|s|$ etc.:

PKE	q	ct size	$\lvert s \rvert$	cutoff gap
NTRU	n	$n \log n$	$n^{1/2}$	$n^{1/2}$
BIKE	2	n	$n^{1/4}$	$n^{1/4}$
McEliece	2	n	$(n/\log n)^{1/2}$	$(\log n)^{1/2}$

Can reduce NTRU gaps by having q grow somewhat more slowly than n; but getting down to a polylog gap requires more powerful decoder, as in McEliece.

(Exercise: GAM/LPR is also $n, n \log n, n^{1/2}, n^{1/2}$.)

So McEliece is NP-hard?

So McEliece is NP-hard?

Fact: **No PKE has ever been proven NP-hard.**

So McEliece is NP-hard?

Fact: **No PKE has ever been proven NP-hard.**

The polylog-gap poly-distance close-vector problem is NP-hard, but this doesn't guarantee security or NP-hardness for the McEliece PKE:

- Maybe it's breakable for *almost all* public keys.
- Maybe it's breakable for public keys that *correspond to McEliece secret keys*.

So McEliece is NP-hard?

Fact: **No PKE has ever been proven NP-hard.**

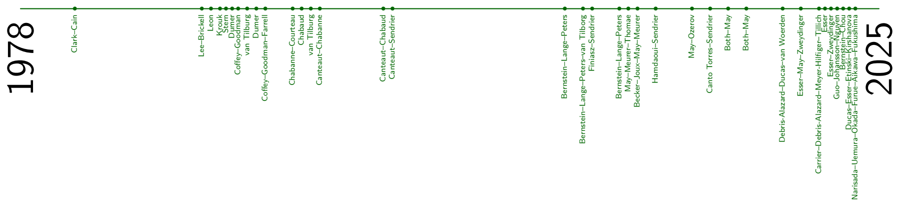
The polylog-gap poly-distance close-vector problem is NP-hard, but this doesn't guarantee security or NP-hardness for the McEliece PKE:

- Maybe it's breakable for *almost all* public keys.
- Maybe it's breakable for public keys that *correspond to McEliece secret keys*.

So the McEliece attack literature studies performance of attacks against uniform random matrices, and studies ways to distinguish Alice's public key from a uniform random matrix.

Stability metric #1: asymptotics

$$\lim_{K\to\infty} \frac{\log_2 \mathrm{AttackCost}_{\mathrm{year}}(K)}{\log_2 \mathrm{AttackCost}_{2025}(K)}$$

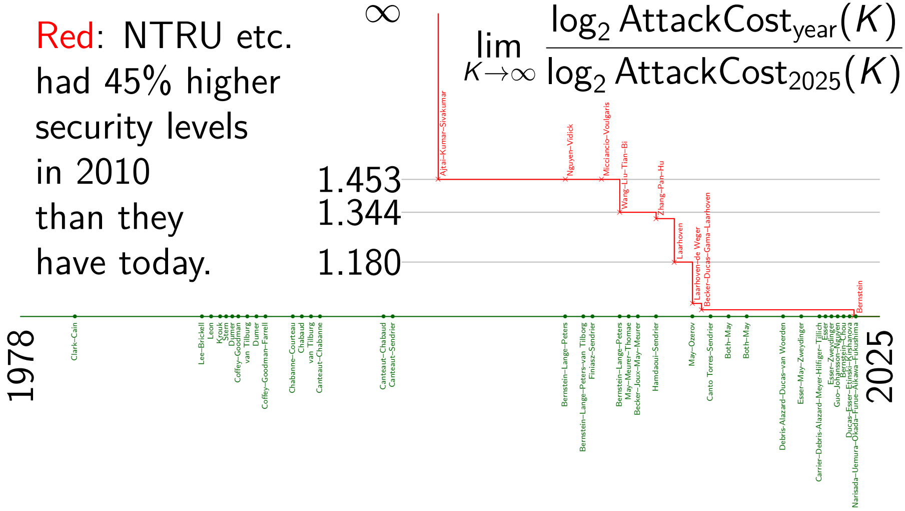


Stability metric #1: asymptotics

Green: McEliece.

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

$$\lim_{K\to\infty} \frac{\log_2 \text{AttackCost}_{\text{year}}(K)}{\log_2 \text{AttackCost}_{2025}(K)}$$



Stability metric #2: challenges

There are scaled-down McEliece challenges: which values of n can academics break? Latest records:

- $n = 1284$ challenge broken as title of a Eurocrypt 2022 paper.
- $n = 1347$ challenge broken using the 2008 Bernstein–Lange–Peters software, **which is as fast as the 2022 software**.
- $n = 1409$ challenge broken on a GPU cluster.

(Exercise: Find large-q attack software from 2008. See how slow it is compared to current software.)

Stability metric #3: bit operations

Crypto 2024 Bernstein–Chou "CryptAttackTester: high-assurance attack analysis": software to

- build complete attack circuits,
- predict circuit cost and probability,
- run small attacks to check accuracy.

Bit operations predicted by CryptAttackTester to attack `mceliece348864` ($n = 3488$):

- $2^{156.96}$: `isd1`, attack ideas from the 1980s.
- $2^{150.59}$: `isd2`, latest attacks.

We also monitor failed attack attempts

Some useless attacks against `mceliece348864`:

- 2000 Sendrier: $>2^{700}$ operations.
- 2011 Faugère–Gauthier–Otmani–Perret–Tillich: inapplicable; also, only a distinguisher.
- 2022 Kirshanova–May: $>2^{9000}$ operations.
- 2023 Mora–Tillich: inapplicable.
- 2023 Couvreur–Mora–Tillich: $>2^{2000}$ operations; also, only a distinguisher.
- 2024 Bardet–Mora–Tillich: inapplicable.
- 2024 Randriambololona: claims 2^{529} operations; also, only a distinguisher.
- 2024 Mora: unclear cost, but much slower than 2000 Sendrier in experiments.

Interlude: “breakthroughs”

“Our work could be a breakthrough”

2002 Courtois–Pieprzyk: “there is a risk that the problem to break Rijndael [i.e., AES] might be subexponential when the number of rounds grows”; this “would be already an important breakthrough”; this is “an important threat for ciphers such as Rijndael, Serpent and Camellia”.

“Our failed attack *is* a breakthrough”

2010 Faugère–Otmani–Perret–Tillich: “To our point of view, disproving/mitigating this hardness assumption is a breakthrough in code-based cryptography and may open a new direction to attack the McEliece cryptosystem.”

“This one is also a breakthrough”

2024 Couvreur–Mora–Tillich: “We introduce a novel algebraic approach for attacking the McEliece cryptosystem which is currently at the 4-th round of the NIST competition . . . This can be considered as a breakthrough . . . 2^{2231}”

The impact of hype

NIST claims that the most recent failed attacks are "significant progress"; claims that these failed attacks "somewhat weaken the argument that the long-term security of Classic McEliece is guaranteed by its long history of cryptanalysis"; and claims that these failed attacks somewhat undermine "the case for treating it as an especially conservative choice".

Look at the numbers

Extensive efforts to break McEliece keep failing: 2^{700}, 2^{2000}, only a distinguisher, 2^{500}, etc.

Cryptanalysis has done much more damage to the security level of large-q systems. Example:

- A 2010 paper proposed lattice dimension **256** for security “about” 2^{150}, “at least” 2^{128}.
- FrodoKEM says it’s an implementation of that paper, but proposes lattice dimension **640** for security 2^{128}.

More security advantages of McEliece

Attacking keys vs. attacking ciphertexts

For each ciphertext size, speed of known attacks:

1. Fastest: Attacking NTRU/LPR/... ciphertexts.
2. Also fastest: Attacking NTRU/LPR/... keys.
3. Much slower: Attacking McEliece ciphertexts.
4. Slowest: Attacking McEliece keys.

Attacking keys vs. attacking ciphertexts

For each ciphertext size, speed of known attacks:

1. Fastest: Attacking NTRU/LPR/... ciphertexts.
2. Also fastest: Attacking NTRU/LPR/... keys.
3. Much slower: Attacking McEliece ciphertexts.
4. Slowest: Attacking McEliece keys.

1+2 exploit weaknesses shared by keys+ciphertexts.

Attacking keys vs. attacking ciphertexts

For each ciphertext size, speed of known attacks:

1. Fastest: Attacking NTRU/LPR/… ciphertexts.
2. Also fastest: Attacking NTRU/LPR/… keys.
3. Much slower: Attacking McEliece ciphertexts.
4. Slowest: Attacking McEliece keys.

1+2 exploit weaknesses shared by keys+ciphertexts.

Some people *praise* this sharing—the supposed simplicity of having just one attack problem. However, a closer look shows that there can be attacks beyond the shared weaknesses; see, e.g., the attack that forced patches in 2023 to FrodoKEM.

The dangers of small keys

BIKE has a “quasi-cyclic” structure:
$K_0, \ldots, K_{n-1}$ are actually
$K, xK, x^2K, \ldots, x^{r-1}K, 1, x, x^2, \ldots, x^{r-1}$
for some public $K \in (\mathbb{Z}/2)[x]/(x^r - 1)$.

The dangers of small keys

BIKE has a “quasi-cyclic” structure: $K_0, \ldots, K_{n-1}$ are actually $K, xK, x^2K, \ldots, x^{r-1}K, 1, x, x^2, \ldots, x^{r-1}$ for some public $K \in (\mathbb{Z}/2)[x]/(x^r - 1)$.

Similar comment applies to NTRU and to many other cola systems, but not McEliece.

The dangers of small keys

BIKE has a “quasi-cyclic” structure:
$K_0, \ldots, K_{n-1}$ are actually
$K, xK, x^2K, \ldots, x^{r-1}K, 1, x, x^2, \ldots, x^{r-1}$
for some public $K \in (\mathbb{Z}/2)[x]/(x^r - 1)$.

Similar comment applies to NTRU and to many other cola systems, but not McEliece.

Why this matters: Some cryptosystems (e.g., the original STOC 2009 Gentry FHE system for cyclotomics) have been broken by attacks exploiting this structure. Crypto 2023 $2^{98.77}$ attack against `bikel1` also exploited this structure.

History: knapsack cryptosystems

1978 Hellman–Merkle: a cryptosystem that uses "trapdoor knapsacks" to hide information.

History: knapsack cryptosystems

1978 Hellman–Merkle: a cryptosystem that uses "trapdoor knapsacks" to hide information.

1982 Shamir, 1983 Adleman, 1983 Brickell–Lagarias–Odlyzko, etc.: breaks of practically all "knapsack" proposals.

History: knapsack cryptosystems

1978 Hellman–Merkle: a cryptosystem that uses “trapdoor knapsacks” to hide information.

1982 Shamir, 1983 Adleman, 1983 Brickell–Lagarias–Odlyzko, etc.: breaks of practically all “knapsack” proposals.

This gave “knapsacks” a very bad reputation. “Lattice-based cryptosystems” are knapsack-based cryptosystems trying to avoid this reputation.

In the meantime: McEliece

1978 McEliece: “A public key cryptosystem based on algebraic coding theory”.

Uses a powerful decoder from 1970 Goppa. We’ll look at this decoder in a moment.

In the meantime: McEliece

1978 McEliece: “A public key cryptosystem based on algebraic coding theory”.

Uses a powerful decoder from 1970 Goppa. We’ll look at this decoder in a moment.

1986 Niederreiter: space improvement, producing the short ciphertexts that I’ve been talking about.

More history: NTRU

1996 Hoffstein–Pipher–Silverman preprint "NTRU: a new high speed public key cryptosystem":

- "In conclusion, for appropriate choice of parameters, NTRU appears to be secure against lattice reduction methods, including any future progress in solving the lattice proximity problem."
- "NTRU bears a superficial resemblance to the McEliece public key cryptosystem."

More history: NTRU

1996 Hoffstein–Pipher–Silverman preprint "NTRU: a new high speed public key cryptosystem":

- "In conclusion, for appropriate choice of parameters, NTRU appears to be secure against lattice reduction methods, including any future progress in solving the lattice proximity problem."
- "NTRU bears a superficial resemblance to the McEliece public key cryptosystem."

1997 Coppersmith–Shamir: better lattice attacks.
1998 Hoffstein–Pipher–Silverman: bigger NTRU.

Perspectives on cola cryptography

2003 Bernstein posting that coined the phrase "post-quantum cryptography" mentioned "lattice-type public-key systems, such as McEliece and NTRU".

2017 Barak similarly summarizes "the 'geometric' or 'coding/lattice'-based systems of the type first proposed by McEliece"—but claims without justification that "known lattice-based public-key encryption schemes can be broken using oracle access to an $O(\sqrt{n})$ approximation algorithm for the lattice closest vector problem". Does "lattice-based" exclude McEliece? Why?

Chosen-ciphertext security

McEliece's original security goal was one-wayness: stopping attacker from finding random s given C.

In 2017, "Classic McEliece" (main focus of current McEliece deployment) converted this into a KEM, adding protection against chosen-ciphertext attacks. QROMCCASecLevel(Classic McEliece) $\geq$ OneWaySecLevel(1978 McEliece) $-$ 5.

Chosen-ciphertext security

McEliece's original security goal was one-wayness: stopping attacker from finding random s given C.

In 2017, "Classic McEliece" (main focus of current McEliece deployment) converted this into a KEM, adding protection against chosen-ciphertext attacks. QROMCCASecLevel(Classic McEliece) $\geq$ OneWaySecLevel(1978 McEliece) $-$ 5.

(GAM/LPR systems such as Kyber and HQC have weaker theorems: chosen-ciphertext security could be 100 bits below one-wayness, or even worse!)

The decoder

How does the decoder work?

Last topic today: Let's look at how Alice decodes $(s_0, \ldots, s_{n-1})$ with high weight (small gap).

How does the decoder work?

Last topic today: Let's look at how Alice decodes $(s_0, \ldots, s_{n-1})$ with high weight (small gap).

System parameters:	Typical
• Integer $m \geq 1$.	$m \in \{12, 13\}$

How does the decoder work?

Last topic today: Let's look at how Alice decodes $(s_0, \ldots, s_{n-1})$ with high weight (small gap).

System parameters:	Typical
• Integer $m \geq 1$.	$m \in \{12, 13\}$
• Integer $n \geq 1$ with $n \leq 2^m$.	$2^{m-1} < n \leq 2^m$

How does the decoder work?

Last topic today: Let's look at how Alice decodes $(s_0, \ldots, s_{n-1})$ with high weight (small gap).

System parameters:	Typical
• Integer $m \ge 1$.	$m \in \{12, 13\}$
• Integer $n \ge 1$ with $n \le 2^m$.	$2^{m-1} < n \le 2^m$
• Integer $w \ge 2$ with $mw < n$.	$w \approx 0.2n/\log_2 n$

How does the decoder work?

Last topic today: Let's look at how Alice decodes $(s_0, \ldots, s_{n-1})$ with high weight (small gap).

System parameters:

	Typical
• Integer $m \geq 1$.	$m \in \{12, 13\}$
• Integer $n \geq 1$ with $n \leq 2^m$.	$2^{m-1} < n \leq 2^m$
• Integer $w \geq 2$ with $mw < n$.	$w \approx 0.2n/\log_2 n$
• Integer $r = mw$.	$r \approx 0.2n$

How does the decoder work?

Last topic today: Let's look at how Alice decodes $(s_0, \ldots, s_{n-1})$ with high weight (small gap).

System parameters:	Typical
• Integer $m \geq 1$.	$m \in \{12, 13\}$
• Integer $n \geq 1$ with $n \leq 2^m$.	$2^{m-1} < n \leq 2^m$
• Integer $w \geq 2$ with $mw < n$.	$w \approx 0.2n/\log_2 n$
• Integer $r = mw$.	$r \approx 0.2n$
• Finite field F with $\#F = 2^m$.	

How does the decoder work?

Last topic today: Let's look at how Alice decodes $(s_0, \ldots, s_{n-1})$ with high weight (small gap).

System parameters:	Typical
• Integer $m \ge 1$.	$m \in \{12, 13\}$
• Integer $n \ge 1$ with $n \le 2^m$.	$2^{m-1} < n \le 2^m$
• Integer $w \ge 2$ with $mw < n$.	$w \approx 0.2n/\log_2 n$
• Integer $r = mw$.	$r \approx 0.2n$
• Finite field F with $\#F = 2^m$.	

For `mceliece348864`: $m = 12$; $n = 3488$; $w = 64$; $r = 768$; $F = (\mathbb{Z}/2)[z]/(z^{12} + z^3 + 1)$.

The McEliece secret key

Alice chooses the following secrets:

- Distinct elements $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ of F.

The McEliece secret key

Alice chooses the following secrets:

- Distinct elements $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ of F.
- Monic irreducible deg-w polynomial $g \in F[x]$: i.e., $g = x^w + g_{w-1}x^{w-1} + \cdots + g_1 x + g_0$, each $g_j \in F$, and g is irreducible in $F[x]$.

The McEliece secret key

Alice chooses the following secrets:

- Distinct elements $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ of F.
- Monic irreducible deg-w polynomial $g \in F[x]$: i.e., $g = x^w + g_{w-1}x^{w-1} + \cdots + g_1 x + g_0$, each $g_j \in F$, and g is irreducible in $F[x]$.

Note that $g(\alpha_i) \neq 0$ since $w \geq 2$.

The McEliece secret key

Alice chooses the following secrets:

- Distinct elements $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ of F.
- Monic irreducible deg-w polynomial $g \in F[x]$: i.e., $g = x^w + g_{w-1}x^{w-1} + \cdots + g_1 x + g_0$, each $g_j \in F$, and g is irreducible in $F[x]$.

Note that $g(\alpha_i) \neq 0$ since $w \geq 2$.

Obvious secret-key format has $(n + w)m$ bits.
There are $(2^m)(2^m - 1) \cdots (2^m - n + 1)$ choices of α, and about $2^{wm}/w$ choices of g.

The McEliece public key

Think of the public key as a linear transformation $H : (\mathbb{Z}/2)^n \to (\mathbb{Z}/2)^{mw}$. Note that everyone can compute the lattice $\{c \in \mathbb{Z}^n : H(c) = 0\}$.

The McEliece public key

Think of the public key as a linear transformation $H : (\mathbb{Z}/2)^n \to (\mathbb{Z}/2)^{mw}$. Note that everyone can compute the lattice $\{c \in \mathbb{Z}^n : H(c) = 0\}$.

Alice chooses a transformation H satisfying the **Goppa property**: $H(c) = 0$ if and only if $\sum_i c_i A/(x - \alpha_i) \in gF[x]$, where $A = \prod_i (x - \alpha_i)$.

The McEliece public key

Think of the public key as a linear transformation $H : (\mathbb{Z}/2)^n \to (\mathbb{Z}/2)^{mw}$. Note that everyone can compute the lattice $\{c \in \mathbb{Z}^n : H(c) = 0\}$.

Alice chooses a transformation H satisfying the **Goppa property**: $H(c) = 0$ if and only if $\sum_i c_i A/(x - \alpha_i) \in gF[x]$, where $A = \prod_i (x - \alpha_i)$.

To avoid revealing any information other than the lattice, Alice chooses H in **systematic form**. This means $H(\text{zeropad}(v)) = v$ for all $v \in (\mathbb{Z}/2)^{mw}$, where $\text{zeropad}(v) = (v, 0, 0, \ldots, 0) \in (\mathbb{Z}/2)^n$.

The decoding algorithm

How Alice decodes a ciphertext:

- Input $C \in (\mathbb{Z}/2)^{mw}$.

The decoding algorithm

How Alice decodes a ciphertext:

- Input $C \in (\mathbb{Z}/2)^{mw}$.
- Interpolate $B \in F[x]$ with $\deg B < n$ and $B(\alpha_i) = \text{zeropad}(C)_i A'(\alpha_i)/g^2(\alpha_i)$ for each i, where A' is the derivative of A.

The decoding algorithm

How Alice decodes a ciphertext:

- Input $C \in (\mathbb{Z}/2)^{mw}$.
- Interpolate $B \in F[x]$ with $\deg B < n$ and $B(\alpha_i) = \text{zeropad}(C)_i A'(\alpha_i)/g^2(\alpha_i)$ for each i, where A' is the derivative of A.
- Compute $a, b \in F[x]$ with $\deg a \le w$, $\deg(aB - bA) < n - w$, and $\gcd\{a, b\} = 1$. (This is a "half-gcd" computation.)

The decoding algorithm

How Alice decodes a ciphertext:

- Input $C \in (\mathbb{Z}/2)^{mw}$.
- Interpolate $B \in F[x]$ with $\deg B < n$ and $B(\alpha_i) = \text{zeropad}(C)_i A'(\alpha_i)/g^2(\alpha_i)$ for each i, where A' is the derivative of A.
- Compute $a, b \in F[x]$ with $\deg a \le w$, $\deg(aB - bA) < n - w$, and $\gcd\{a, b\} = 1$. (This is a "half-gcd" computation.)
- Compute $s \in (\mathbb{Z}/2)^n$ with $s_i = [a(\alpha_i) = 0]$, i.e., $s_i = 1$ if and only if $a(\alpha_i) = 0$.
- Output s.

Magic fact: The algorithm works

Fact: If $s \in (\mathbb{Z}/2)^n$ has weight w and $C = H(s)$ then the algorithm outputs s.

Converse: If the algorithm outputs $s \in (\mathbb{Z}/2)^n$ and s has weight w then $C = H(s)$.

To understand *why* this works,
take a course on coding theory,
or read my minicourse on this algorithm:
`cr.yp.to/papers.html#goppadecoding`.