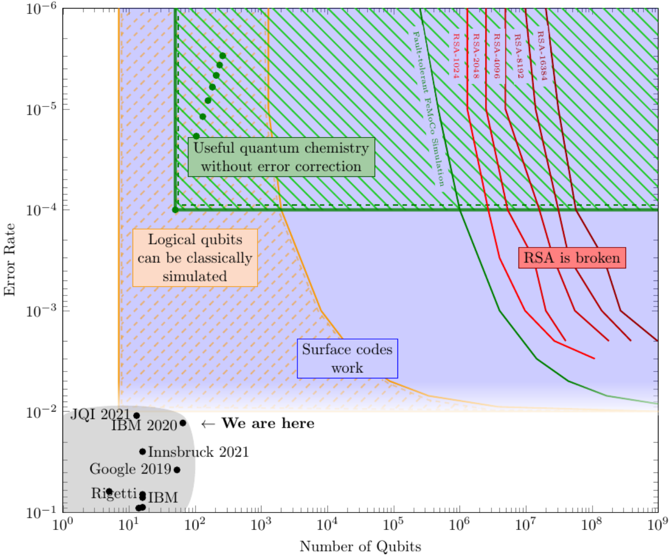
Post-quantum cryptography for developers

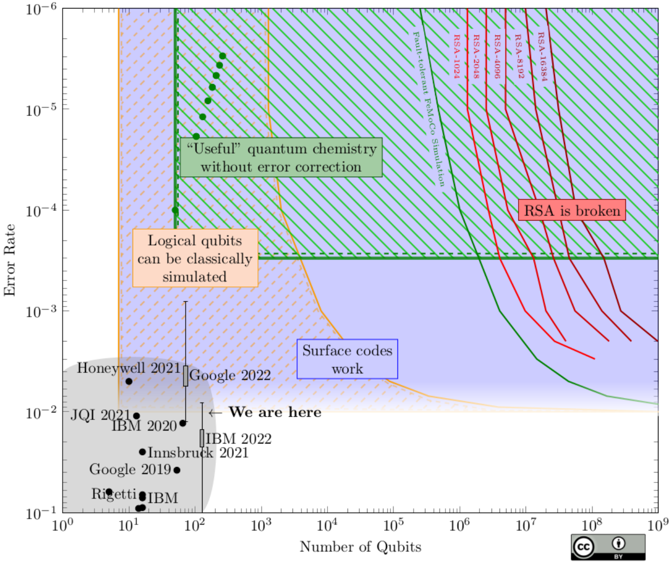
Daniel J. Bernstein

Do applications have to worry about quantum computers?

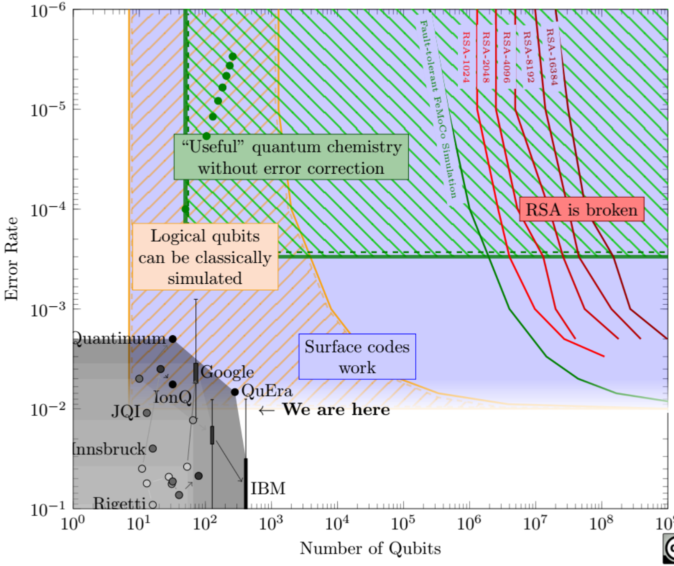
2021 Jaques: Quantum Landscape



2022 Jaques: Quantum Landscape



2023 Jaques: Quantum Landscape



Computers → disaster ← algorithms

Graph shows quantum computers advancing on the critical axes: #qubits, error rate.

Computers ➔ disaster ← algorithms

Graph shows quantum computers advancing on the critical axes: #qubits, error rate.

2023.01 Bernstein: “It’s fascinating to see how the historical data in the bottom-left corner [of that graph] leads readers to guess the number of years to the top right without realizing that the top right is a moving target.”

Computers ➔ disaster ← algorithms

Graph shows quantum computers advancing on the critical axes: #qubits, error rate.

2023.01 Bernstein: “It’s fascinating to see how the historical data in the bottom-left corner [of that graph] leads readers to guess the number of years to the top right without realizing that the top right is a moving target.”

2023.01 Jaques: “Thanks, I guess? I don’t expect much movement in those lines so I’m glad the diagram conveys that”

Computers→disaster←algorithms

Graph shows quantum computers advancing on the critical axes: #qubits, error rate.

2023.01 Bernstein: “It’s fascinating to see how the historical data in the bottom-left corner [of that graph] leads readers to guess the number of years to the top right without realizing that the top right is a moving target.”

2023.01 Jaques: “Thanks, I guess? I don’t expect much movement in those lines so I’m glad the diagram conveys that”

2023.08 Regev: improvement to Shor’s algorithm.

Computers ➔ disaster ⟵ algorithms

Graph shows quantum computers advancing on the critical axes: #qubits, error rate.

2023.01 Bernstein: "It's fascinating to see how the historical data in the bottom-left corner [of that graph] leads readers to guess the number of years to the top right without realizing that the top right is a moving target."

2023.01 Jaques: "Thanks, I guess? I don't expect much movement in those lines so I'm glad the diagram conveys that"

2023.08 Regev: improvement to Shor's algorithm. 2023.12 Jaques: "Whoops!"

So: when will RSA-2048 be broken?

Global Risk Institute 2023 survey of 37 people working on quantum computing shows wide range of predictions regarding chance of RSA-2048 break by year Y.

e.g. $Y = 2038$: 6 say $>$95%; 4 say $>$70%; 10 say ~50%; 10 say $<$30%; 7 say $<$5%.

So: when will RSA-2048 be broken?

Global Risk Institute 2023 survey of 37 people working on quantum computing shows wide range of predictions regarding chance of RSA-2048 break by year Y.
e.g. $Y = 2038$: 6 say $>$95%; 4 say $>$70%; 10 say ~50%; 10 say $<$30%; 7 say $<$5%.

My assessment: reaches 50% by $Y = 2029$; 50% for public demonstration by $Y = 2032$. Note that attackers are ahead of the public.

So: when will RSA-2048 be broken?

Global Risk Institute 2023 survey of 37 people working on quantum computing shows wide range of predictions regarding chance of RSA-2048 break by year Y.
e.g. $Y = 2038$: 6 say $>$95%; 4 say $>$70%; 10 say ~50%; 10 say $<$30%; 7 say $<$5%.

My assessment: reaches 50% by $Y = 2029$; 50% for public demonstration by $Y = 2032$. Note that attackers are ahead of the public.

“You always say it’s 10 years away!”

So: when will RSA-2048 be broken?

Global Risk Institute 2023 survey of 37 people working on quantum computing shows wide range of predictions regarding chance of RSA-2048 break by year Y.
e.g. $Y = 2038$: 6 say $>$95%; 4 say $>$70%;
10 say ~50%; 10 say $<$30%; 7 say $<$5%.

My assessment: reaches 50% by $Y = 2029$;
50% for public demonstration by $Y = 2032$.
Note that attackers are ahead of the public.

“You always say it’s 10 years away!”—No.
See the bets I placed in 2014, 2017, and 2023.

Reasons to take action right now

I *hope* quantum computing somehow fails. But if it works then it breaks RSA-2048 and ECC-256. “Wait and see” isn’t safe:

- Attackers are already recording ciphertexts today to decrypt later.

Reasons to take action right now

I *hope* quantum computing somehow fails.
But if it works then it breaks RSA-2048 and ECC-256. “Wait and see” isn’t safe:

- Attackers are already recording ciphertexts today to decrypt later.
- Some environments are very slow to roll out new software for encryption+signatures.

Reasons to take action right now

I *hope* quantum computing somehow fails.
But if it works then it breaks RSA-2048 and ECC-256. “Wait and see” isn’t safe:

- Attackers are already recording ciphertexts today to decrypt later.
- Some environments are very slow to roll out new software for encryption+signatures.
- We’ll need even more time to fix whatever screwups happen.

Which post-quantum primitives should we implement?

Trust the NIST competition?

Out of the 69 round-1 submissions to the competition in 2017: **48%** are broken by now.

(AES-128 was the minimum security level allowed in the competition. Broken means: smallest proposed parameters are now known to be easier to break than AES-128.)

Trust the NIST competition?

Out of the 69 round-1 submissions to the competition in 2017: **48%** are broken by now.

(AES-128 was the minimum security level allowed in the competition. Broken means: smallest proposed parameters are now known to be easier to break than AES-128.)

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

Trust the NIST competition?

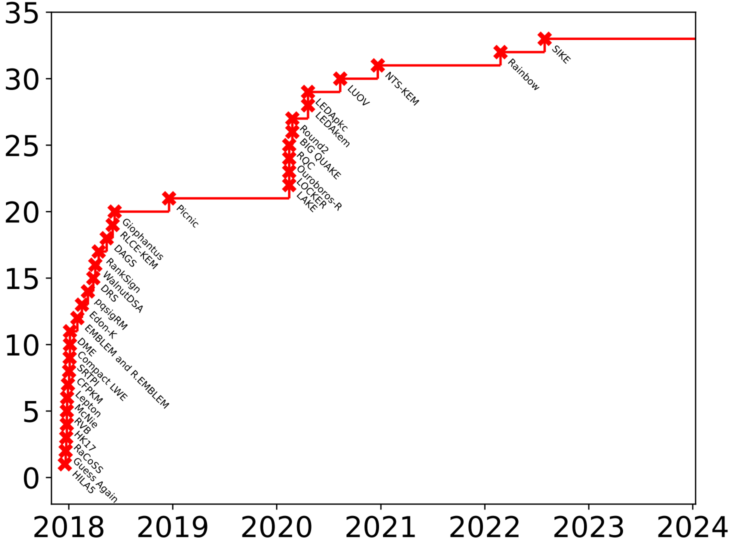
Out of the 69 round-1 submissions to the competition in 2017: **48%** are broken by now.

(AES-128 was the minimum security level allowed in the competition. Broken means: smallest proposed parameters are now known to be easier to break than AES-128.)

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

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

When the breaks were published



Always wear your seatbelt

SIKE: the last break ever?

Always wear your seatbelt

SIKE: the last break ever? I doubt it.

Always wear your seatbelt

SIKE: the last break ever? I doubt it.
e.g. security levels for lattices keep dropping.

Always wear your seatbelt

SIKE: the last break ever? I doubt it.
e.g. security levels for lattices keep dropping.

NSA says throw away ECC. Ignore them.
Encrypt with post-quantum system *and* ECC.

Always wear your seatbelt

SIKE: the last break ever? I doubt it.
e.g. security levels for lattices keep dropping.

NSA says throw away ECC. Ignore them.
Encrypt with post-quantum system *and* ECC.

e.g. 2019 Google–Cloudflare experiments:
CECPQ2 used `ntruhrss701` + X25519;
CECPQ2b used `sikep434` (broken!) + X25519.

Always wear your seatbelt

SIKE: the last break ever? I doubt it.
e.g. security levels for lattices keep dropping.

NSA says throw away ECC. Ignore them.
Encrypt with post-quantum system *and* ECC.

e.g. 2019 Google–Cloudflare experiments:
CECPQ2 used `ntruhrss701` + X25519;
CECPQ2b used `sikep434` (broken!) + X25519.
e.g. OpenSSH uses `sntrup761` + X25519.

Always wear your seatbelt

SIKE: the last break ever? I doubt it.
e.g. security levels for lattices keep dropping.

NSA says throw away ECC. Ignore them.
Encrypt with post-quantum system *and* ECC.

e.g. 2019 Google–Cloudflare experiments:
CECPQ2 used `ntruhrss701` + X25519;
CECPQ2b used `sikep434` (broken!) + X25519.
e.g. OpenSSH uses `sntrup761` + X25519.
e.g. Google uses `ntruhrss701` + X25519
for its internal communication.

Always wear your seatbelt

SIKE: the last break ever? I doubt it.
e.g. security levels for lattices keep dropping.

NSA says throw away ECC. Ignore them.
Encrypt with post-quantum system *and* ECC.

e.g. 2019 Google–Cloudflare experiments:
CECPQ2 used `ntruhrss701` + X25519;
CECPQ2b used `sikep434` (broken!) + X25519.
e.g. OpenSSH uses `sntrup761` + X25519.
e.g. Google uses `ntruhrss701` + X25519
for its internal communication.
e.g. Chrome supports `kyber768` + X25519.

Also: try not to crash the car

1978 McEliece system: strongest security track record of all post-quantum proposals.

Long-term security: `mceliece6688128`.
Bytes in pk, sk, ct: 1044992, 13932, 208.

Also: try not to crash the car

1978 McEliece system: strongest security track record of all post-quantum proposals.

Long-term security: `mceliece6688128`. Bytes in pk, sk, ct: 1044992, 13932, 208.

If application can't handle the pk size, then use lattices, but cautiously: e.g., McEliece for identity keys + lattices for forward secrecy + ECC so you're definitely not losing security.

Use biggest available lattice dimensions to leave a security margin: `ntruhps40961229`, `sntrup1277`, `frodokem1344`, `ntruhrss1373`.

Trigger warning:

Trigger warning:
PATENTS

If you find patents traumatic,
or if you have a policy to not learn about patents,
please skip the next slide.

Lattice patents

Google Patents finds, e.g., 3997 results for "post-quantum". Mostly uninteresting, but we know some examples of problems. Patent analysis has not been systematic.

No known issues for `ntruhps`, `ntruhrss`, `sntrup`. Basic NTRU patent expired in 2017.

NIST says it has a license for 2010 GAM patent (LPR) and 2012 Ding patent if you use exactly the Kyber standard. Those patents still cover newhope, `saber`, `ntrulpr`, etc.

Zhao says "Kyber is covered by our patents".

Where can we find code to reuse?

Sources of post-quantum software

Design teams typically submitted a “portable” reference implementation and a faster Intel/AMD implementation to SUPERCOP. Usually teams maintain web pages, often pointing to further implementations.

Most implementations today are repackaged copies of those, sometimes translated into other languages. PQClean and liboqs include tweaks for MSVC portability.

I’m optimistic about future of microlibraries such as libmceliece (and lib25519).

**What should we
watch out for in implementing
post-quantum primitives?**

Randomness is used in tricky ways

The bug was in the function `rej_gamma1m1` *in* `poly.c` *and consisted of accidentally overwriting a variable prior to using it. . . . the result of the bug was that the same randomness ended up being used for pairs of consecutive coefficients . . . This reuse of randomness can easily be exploited to recover the secret key and we thus emphasize that the software, in the state submitted to NIST, should not be used in any real application.*

—Dilithium comment, January 2018

And other tricky ways

. . . the new Falcon implementation published on 2019-08-02 had two severe bugs in the sampler; one table was wrong, and a scaling factor was applied at the wrong place. . . . Produced signatures were valid but leaked information on the private key. . . . The fact that these bugs existed in the first place shows that the traditional development methodology (i.e. 'being super careful') has failed.

—Falcon comment, September 2019

Timing leaks from, e.g., `memcmp`

Experiments show that the attack code is able to extract the secret key for all security levels using about 2^{30} decapsulation calls.

—FrodoKEM attack paper ("A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM"), June 2020

Misimplementing `memcmp`

It looks like the FrodoKEM team also fixed the timing oracle [GJN20] badly and caused a more serious security problem while trying to do that. … A decryption failure is very likely to be ignored by this rejection mechanism because the selector will be 0 with a high probability even in case of a mismatch, the failed decryption will be used and returned to the caller … I'd assume that the key recovery attacks of [GJN20] are even easier to mount with this new, more powerful oracle.

—FrodoKEM comment, December 2020

KyberSlash1

I noticed various `"/KYBER_Q"` *occurrences with variable inputs. In at least one case, line 190 of* `crypto_kem/kyber768/ref/poly.c`*, this is clearly a secret input. I'd expect measurable, possibly exploitable, timing variations … [Maybe compilers] convert divisions to multiplications, but this depends very much on compiler options. Within available tools to scan for variable-time instructions, a few (e.g., saferewrite) know how to check for divisions but most don't.*

—Kyber comment (from me), December 2023; see also my subsequent attack demo

KyberSlash2

During our analysis, we stumbled upon another another source of timing variability in several patched implementations of Kyber . . . We believe, the aforementioned division operations were not considered to be problematic since they were present in the encryption procedure whose ciphertext output is considered to be public. However, the encryption procedure is also used for re-encryption in the FO transformed decapsulation procedure . . .

—Kyber comment, December 2023

Another reason for seatbelts

KyberSlash: the last *implementation* break ever?

Another reason for seatbelts

KyberSlash: the last *implementation* break ever? I very much doubt it.

Another reason for seatbelts

KyberSlash: the last *implementation* break ever? I very much doubt it.

Implementing post-quantum primitives is more complex than implementing ECC, and the community has many years fewer experience with what goes wrong.

Surely many traps remain to be discovered.

Another reason for seatbelts

KyberSlash: the last *implementation* break ever? I very much doubt it.

Implementing post-quantum primitives is more complex than implementing ECC, and the community has many years fewer experience with what goes wrong.

Surely many traps remain to be discovered.

Some tools that I recommend: negative tests; known-randomness tests; timing-variability scanners; tools for formal verification.