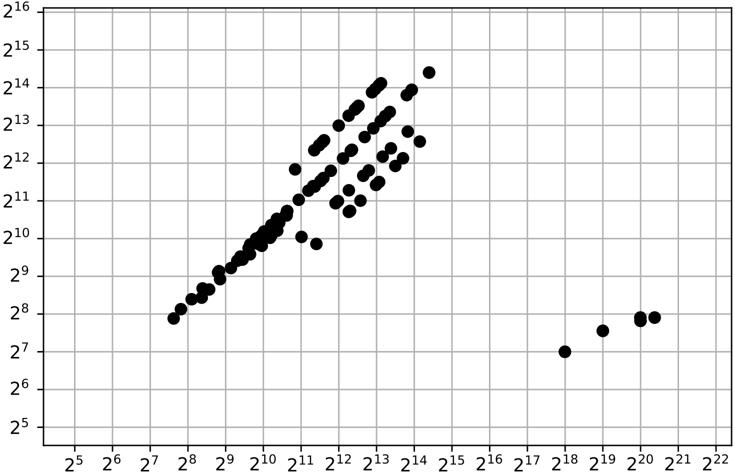
How to fit an elephant into a Smart car – PQC for small devices

Tanja Lange

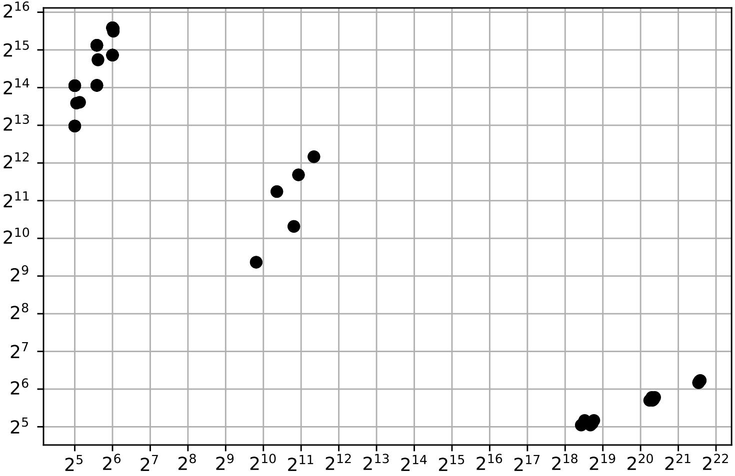
Eindhoven University of Technology & Academia Sinica

Encryption (KEM): ciphertext size (vertical) vs. public-key size (horizontal)



For more graphs incl. speed comparison on many CPUs see
`http://bench.cr.yp.to/results-kem.html`.
Graphs linked with every CPU.

Signatures: signature size (vertical) vs. public-key size (horizontal)



For more graphs incl. speed comparison on many CPUs see
http://bench.cr.yp.to/results-sign.html.
Graphs linked with every CPU.

Verifying Post-Quantum Signatures in 8 kB of RAM

Gonzalez, Hülsing, Kannwischer, Krämer, Lange, Stöttinger, Waitz, Wiggers, Yang,
https://eprint.iacr.org/2021/662, https://git.fslab.de/pqc/streaming-pq-sigs

- Setup: small processor with limited memory.
  We used ARM Cortex-M3, restricted the available RAM to 8 kB.
- The board did provide 8 kB additional flash storage.

Verifying Post-Quantum Signatures in 8 kB of RAM

Gonzalez, Hülsing, Kannwischer, Krämer, Lange, Stöttinger, Waitz, Wiggers, Yang,
`https://eprint.iacr.org/2021/662`, `https://git.fslab.de/pqc/streaming-pq-sigs`

- Setup: small processor with limited memory.
  We used ARM Cortex-M3, restricted the available RAM to 8 kB.
- The board did provide 8 kB additional flash storage.
- General assumption for streaming is some nearby storage.
- This storage is typically untrusted
  - May not hold a private key,
  - may not decide validity of signatures, ...
  - Integrity of steamed in data must be tested.
- Despite these limitations, this extra storage is helpful (and often necessary).
- For some systems, 8 kB is not enough to hold the key, for some not enough to hold the signature, for some not enough to hold the optimized implementations.

Table: Communication overhead in bytes and milliseconds at 500 kbit/s and 20 Mbit/s. GeMSS requires to stream in the public key *nb_ite* times (4 for `gemss-128`). All other schemes require streaming in the public key and signed message once.

	streaming data			streaming time	
	$\|pk\|$	$\|sig\|$	total	500 kbit/s	20 Mbit/s
`sphincs-s`[a]	32	7 856	7 888	126.2 ms	3.2 ms
`sphincs-f`[b]	32	17 088	17 120	273.9 ms	6.9 ms
`rainbowI-classic`	161 600	66	161 666	2 586.7 ms	64.7 ms
`gemss-128`	352 188	33	1 408 785[c]	22 540.6 ms	563.5 ms
`dilithium2`	1 312	2 420	3 732	59.7 ms	1.5 ms
`falcon-512`	897	690	1 587	25.4 ms	0.6 ms

[a] `-sha256-128s-simple` [b] `-sha256-128f-simple` [c] $4 \cdot |pk| + |sig|$

Table: Cycle count for signature verification for a 33-byte message. Average over 1 000 signature verifications. Hashing cycles needed for verification of the streamed in public key (hashing and comparing to embedded hash) are reported separately. We also report the verification time on a practical HSM running at 100 MHz and also the total time including the streaming at 20 Mbit/s.

	w/o pk vrf.	w/ pk verification			w/ streaming
		pk vrf.	total	time[e]	20 Mbit/s
sphincs-s[a]	8 741k	0	8 741k	87.4 ms	90.6 ms
sphincs-f[b]	26 186k	0	26 186k	261.9 ms	268.7 ms
rainbowI-classic	333k	6 850k[d]	7 182k	71.8 ms	136.5 ms
gemss-128	1 619k	109 938k[c]	111 557k	1 115.6 ms	1 679.1 ms
dilithium2	1 990k	133k[c]	2 123k	21.2 ms	21.8 ms
falcon-512	581k	91k[c]	672k	6.7 ms	8.2 ms

[a] -sha256-128s-simple [b] -sha256-128f-simple [c] SHA-3/SHAKE [d] SHA-256
[e] At 100 MHz (no wait states)

Table: Memory and code-size requirements in bytes for our implementations. Memory includes stack needed for computations, global variables stored in the .bss section and the buffer required for streaming. Code-size excludes platform and framework code as well as code for `SHA-256` and `SHA-3`.

	memory				code
	total	buffer	.bss	stack	.text
`sphincs-s`[a]	6 904	4 928	780	1 196	2 724
`sphincs-f`[b]	7 536	4 864	780	1 892	2 586
`rainbowI-classic`	8 168	6 848	724	596	2 194
`gemss-128`	8 176	4 560	496	3 120	4 740
`dilithium2`	8 048	40	6 352	1 656	7 940
`falcon-512`	6 552	897	5 255	400	5 784

[a] `-sha256-128s-simple` [b] `-sha256-128f-simple`

NIST PQC submission Classic McEliece

No patents.
Shortest ciphertexts.
Fast open-source constant-time software implementations.
Very conservative system, expected to last; has strongest security track record.

Sizes with similar post-quantum security to AES-128, AES-192, AES-256:

Metric	**mceliece348864**	**mceliece460896**	**mceliece6960119**
Public-key size	261120 bytes	524160 bytes	1047319 bytes
Secret-key size	6452 bytes	13568 bytes	13908 bytes
Ciphertext size	128 bytes	188 bytes	226 bytes
Key-generation time	52415436 cycles	181063400 cycles	417271280 cycles
Encapsulation time	43648 cycles	77380 cycles	143908 cycles
Decapsulation time	130944 cycles	267828 cycles	295628 cycles

See `https://classic.mceliece.org` for authors, details & parameters.

Optimized implementations for Cortex-M4

pqm4, 2019: Classic McEliece public keys are “too large to fit into the memory of our platform”

Optimized implementations for Cortex-M4

pqm4, 2019: Classic McEliece public keys are "too large to fit into the memory of our platform"

Classic McEliece implementation with low memory footprint (Roth, Karatsiolis, Krämer; CARDIS 2020). "an implementation of Classic McEliece on an ARM Cortex-M4 processor, optimized to overcome memory constraints"; stream public key off device

Classic McEliece on the ARM Cortex-M4 (Chen, Chou; CHES 2021). `mceliece348864` fits on Cortex-M4, including public key!

- 2 146 932 033 keygen (only 1 430 811 294 for `f` version).
- 582 199 encap
- 2 706 681 decap

`mceliece8192128`: 7 481 747 for decap (private keys are tiny).

Small ciphertext makes a large difference

PQ-WireGuard (Hülsing, Ning, Schwabe, Weber, Zimmermann; IEEE S&P 2021).

- Uses McEliece for long-term identity key in KEM-KEM construction.
- McEliece key exchanged out of band at registration.
- Strong benefit from short ciphertexts.
- Combined with lattice-based scheme for ephemeral keys.

McTiny (Bernstein, Lange; USENIX Security 2020)

- McEliece also used for ephemeral keys.
- Avoids DoS memory flooding attacks by using structure of code-based encryption. Server returns partial encryption and state in cookie encrypted to itself; cookie is smaller than network packet sent to server.
- Good speed and security with congestion control and surrounding protocol.

Different deployment strategy

PQConnect: An Automated Boring Protocol for Quantum-Secure Tunnels

- Do not patch PQC onto existing network protocols, but add a new layer with superior security.

Different deployment strategy

PQConnect: An Automated Boring Protocol for Quantum-Secure Tunnels

- Do not patch PQC onto existing network protocols, but add a new layer with superior security.
- Can be gradually deployed.
- Add support for VPN-like tunnels to clients and servers

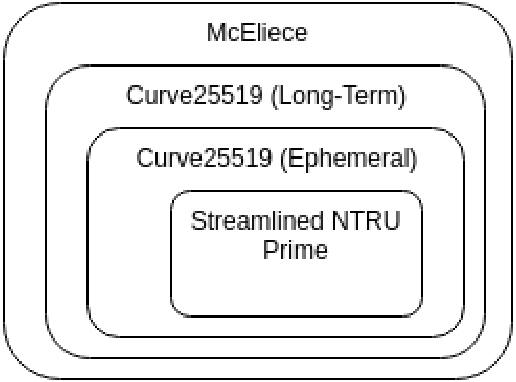
Different deployment strategy

PQConnect: An Automated Boring Protocol for Quantum-Secure Tunnels

- Do not patch PQC onto existing network protocols, but add a new layer with superior security.
- Can be gradually deployed.
- Add support for VPN-like tunnels to clients and servers but do this to the endpoints, not some intermediate VPN server.
- PQConnect is designed for security, handshake and ratcheting proven using Tamarin prover (formal verification tool).
- Use Curve25519 (pre-quantum) and Classic McEliece (conservative PQC) for long-term identity keys.
- Use Curve25519 (pre-quantum) and lattice-based Streamlined NTRU Prime (PQC) for ephemeral keys.

PQConnect handshake: Nesting schemes

Most conservative system on the outside.



Attacker can see long-term Curve25519 identity key,
can break it with a quantum computer,
but cannot obtain DH value as client's share is wrapped.

PQConnect handshake: Handling McElice keys

- McEliece is used for the long-term key, i.e., this key does not change.
- Store key for frequently visited sites (Google, Gmail, Facebook, Twitter,. . .)
- Link key download to obtaining IP address via DNS lookup.
  This is how the client know where to connect to. PQConnect piggy-backs on this with a hash of the key and info on where to download the key.

PQConnect handshake: Handling McElice keys

- McEliece is used for the long-term key, i.e., this key does not change.
- Store key for frequently visited sites (Google, Gmail, Facebook, Twitter,...)
- Link key download to obtaining IP address via DNS lookup.
  This is how the client know where to connect to. PQConnect piggy-backs on this with a hash of the key and info on where to download the key.
- Split key as in McTiny, download in small chunks and verify with hash;
  PQConnect also includes the Curve25519 key (256 bits, just a small corner).
- PQConnect benefits from small McEliece ciphertexts.
- Combine with lattice-based crypto for balance in ciphertext and public key size; security concerns alleviated by nesting.
- More information on protocol:
  `https://research.tue.nl/en/studentTheses/pqconnect`
  Paper and software still forthcoming.

Protective measures for pre-quantum cryptography

Aka poor-man's PQC

Premise: Known/slowly changing set of peers. Does not fit web servers.
This fits email, messaging (Signal, etc.), most enterprise setups.

Protective measures for pre-quantum cryptography

Aka poor-man's PQC

Premise: Known/slowly changing set of peers. Does not fit web servers.
This fits email, messaging (Signal, etc.), most enterprise setups.

Requires: Adjust protocol to have user keep state per peer.

Protective measures for pre-quantum cryptography

Aka poor-man's PQC

Premise: Known/slowly changing set of peers. Does not fit web servers.
This fits email, messaging (Signal, etc.), most enterprise setups.

Requires: Adjust protocol to have user keep state per peer.

Option 1: Have fixed secret per peer, include this in KDF.
Secret exchanged out of band, or exchange is not observed.
Provided in WireGuard as option.

Protective measures for pre-quantum cryptography

Aka poor-man's PQC

Premise: Known/slowly changing set of peers. Does not fit web servers.
This fits email, messaging (Signal, etc.), most enterprise setups.

Requires: Adjust protocol to have user keep state per peer.

Option 1: Have fixed secret per peer, include this in KDF.
Secret exchanged out of band, or exchange is not observed.
Provided in WireGuard as option.

Option 2: Have updatable secret per peer, include this in KDF.
Update per-peer secret with each new public-key operation.
Initial secret exchanged out of band, or exchange is not observed.

Details worked out in RFC 6189 on ZRTP, see also section 6.2 of the ENISA report.

Use 256-bit keys for AES or ChaCha20 (good idea anyways).
No need to change MAC lengths for information-theoretic MACs
(Wegman-Carter, such as GMAC & Poly1305).

Further information

- YouTube channel Tanja Lange: Post-quantum cryptography.
- https://2017.pqcrypto.org/school: PQCRYPTO summer school with 21 lectures on video, slides, and exercises.
- https://2017.pqcrypto.org/exec, https://pqcschool.org/index.html: Executive schools (less math, more perspective).
- https://pqcrypto.org our overview page.
- ENISA report on PQC, co-authored.
- https://pqcrypto.eu.org: PQCRYPTO EU Project.
  - PQCRYPTO recommendations.
  - Free software libraries (libpqcrypto, pqm4, pqhw).
  - Many reports, scientific articles, (overview) talks.
- Quantum Threat Timeline from Global Risk Institute, 2019; 2021 update.
- Status of quantum computer development (by German BSI).
- NIST PQC competition.
- PQCrypto 2016, PQCrypto 2017, PQCrypto 2018, PQCrypto 2019, PQCrypto 2020, PQCrypto 2021 with many slides and videos online.