Classic McEliece:
conservative code-based cryptography

Daniel J. Bernstein\textsuperscript{1}, Tung Chou\textsuperscript{2}, Tanja Lange\textsuperscript{3},
Ingo von Maurich, Rafael Misoczki\textsuperscript{4}, Ruben Niederhagen\textsuperscript{5},
Edoardo Persichetti\textsuperscript{6}, Christiane Peters, Peter Schwabe\textsuperscript{7},
Nicolas Sendrier\textsuperscript{8}, Jakub Szefer\textsuperscript{9}, Wen Wang\textsuperscript{9}

\textsuperscript{1}University of Illinois at Chicago, \textsuperscript{2}Osaka University,
\textsuperscript{3}Technische Universiteit Eindhoven, \textsuperscript{4}Intel Corporation,
\textsuperscript{5}Fraunhofer SIT, \textsuperscript{6}Florida Atlantic University,
\textsuperscript{7}Radboud University, \textsuperscript{8}Inria, \textsuperscript{9}Yale University

12 April 2018
First NIST PQC workshop
Key sizes and key-generation speed

mceliece6960119 parameter set:
  1047319 bytes for public key.
  13908 bytes for secret key.

mceliece8192128 parameter set:
  1357824 bytes for public key.
  14080 bytes for secret key.
Key sizes and key-generation speed

mceliece6960119 parameter set:
  1047319 bytes for public key.
  13908 bytes for secret key.

mceliece8192128 parameter set:
  1357824 bytes for public key.
  14080 bytes for secret key.

Current software: billions of cycles to generate a key; not much optimization effort yet.
All code runs in constant time.
Key sizes and key-generation speed

mceliece6960119 parameter set:
  1047319 bytes for public key.
  13908 bytes for secret key.

mceliece8192128 parameter set:
  1357824 bytes for public key.
  14080 bytes for secret key.

Current software: billions of cycles to generate a key; not much optimization effort yet.
All code runs in constant time.

Very fast in hardware (PQCrypto 2018; CHES 2017):
a few million cycles at 231MHz
using 129059 modules, 1126 RAM blocks
on Altera Stratix V FPGA.

Classic McEliece

https://classic.mceliece.org/
Short ciphertexts

mceliece6960119 parameter set:
  226 bytes for ciphertext.

mceliece8192128 parameter set:
  240 bytes for ciphertext.
Short ciphertexts

mceliece6960119 parameter set:
   226 bytes for ciphertext.

mceliece8192128 parameter set:
   240 bytes for ciphertext.

Constant time software (measured on Haswell, larger parameters):
   295932 cycles for enc,
   355152 cycles for dec (decoding, hashing, etc.).
Short ciphertexts

mceliece6960119 parameter set:
  226 bytes for ciphertext.

mceliece8192128 parameter set:
  240 bytes for ciphertext.

Constant time software (measured on Haswell, larger parameters):
  295932 cycles for enc,
  355152 cycles for dec (decoding, hashing, etc.).

Again very fast in hardware:
  17140 cycles for decoding.
Short ciphertexts

mceliece6960119 parameter set:
   226 bytes for ciphertext.

mceliece8192128 parameter set:
   240 bytes for ciphertext.

Constant time software (measured on Haswell, larger parameters):
   295932 cycles for enc,
   355152 cycles for dec (decoding, hashing, etc.).

Again very fast in hardware:
   17140 cycles for decoding.

Can tweak parameters for even smaller ciphertexts, not much penalty in key size.

Classic McEliece
https://classic.mceliece.org/
One-wayness (OW-CPA)

Fundamental security question:
Given random parity-check matrix $H$ and syndrome $s$, can attacker efficiently find $e$ with $s = He$?
One-wayness (OW-CPA)

Fundamental security question:
Given random parity-check matrix $H$ and syndrome $s$, can attacker efficiently find $e$ with $s = He$?

1962 Prange: simple attack idea
guiding sizes in 1978 McEliece.
One-wayness (OW-CPA)

Fundamental security question:
Given random parity-check matrix $H$ and syndrome $s$, can attacker efficiently find $e$ with $s = He$?


The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(lg \lambda)^2$-bit keys as $\lambda \to \infty$ to achieve $2^\lambda$ security against Prange’s attack.
Here $c_0 \approx 0.7418860694$. 

Classic McEliece

https://classic.mceliece.org/
40 years and more than 30 analysis papers later


The McEliece system uses ($c_0 + o(1)$) $\lambda^2 (\log \lambda)^2$-bit keys as $\lambda \to \infty$ to achieve $2^\lambda$ security against all attacks known today. Same $c_0 \approx 0.7418860694$. Replacing $\lambda$ with $2^\lambda$ stops all known quantum attacks.

Classic McEliece

https://classic.mceliece.org/
40 years and more than 30 analysis papers later


The McEliece system uses \((c_0 + o(1))\lambda^2(\lg \lambda)^2\)-bit keys as \(\lambda \to \infty\) to achieve \(2^\lambda\) security against all attacks known today. Same \(c_0 \approx 0.7418860694\).
40 years and more than 30 analysis papers later


The McEliece system uses \((c_0 + o(1))\lambda^2(\lg \lambda)^2\)-bit keys as \(\lambda \to \infty\) to achieve \(2^\lambda\) security against all attacks known today. Same \(c_0 \approx 0.7418860694\).

Replacing \(\lambda\) with \(2\lambda\) stops all known quantum attacks.
Classic McEliece

McEliece’s system prompted huge amount of followup work.

Some work improves efficiency while clearly preserving security:

- Niederreiter’s dual PKE
  (use parity check matrix instead of generator matrix);
- many decoding speedups; . . .

Classic McEliece uses all this, with constant-time implementations.

- Write $H = (I_{n-k} | T)$, public key is $(n - k) \times k$ matrix $T$, $n - k = w \log_2 q$. $H$ constructed from binary Goppa code.
- Encapsulate using $e$ of weight $w$.

$q = 8192, n = 6960, w = 119$.

`mceliece8192128` parameter set:
$q = 8192, n = 8192, w = 128$.
IND-CCA2 conversions

Classic McEliece follows best practices from literature:

1. Session key: feed random $e$ through standard hash function.
IND-CCA2 conversions

Classic McEliece follows best practices from literature:
1. Session key: feed random $e$ through standard hash function.
2. Ciphertext includes another hash of $e$ ("confirmation").
IND-CCA2 conversions

Classic McEliece follows best practices from literature:
1. Session key: feed random $e$ through standard hash function.
2. Ciphertext includes another hash of $e$ (“confirmation”).
3. Dec includes recomputation and verification of ciphertext.

Classic McEliece
https://classic.mceliece.org/
Classic McEliece follows best practices from literature:

1. Session key: feed random $e$ through standard hash function.
2. Ciphertext includes another hash of $e$ (“confirmation”).
3. Dec includes recomputation and verification of ciphertext.
4. KEM never fails: if inversion fails or ciphertext does not match, return hash of (secret, ciphertext).

Further features of system that simplify attack analysis:

5. Ciphertext is deterministic function of input $e$: i.e., inversion recovers all randomness used to create ciphertexts.
6. There are no inversion failures for legitimate ciphertexts.
IND-CCA2 conversions

Classic McEliee follows best practices from literature:

1. Session key: feed random $e$ through standard hash function.
2. Ciphertext includes another hash of $e$ ("confirmation").
3. Dec includes recomputation and verification of ciphertext.
4. KEM never fails: if inversion fails or ciphertext does not match, return hash of (secret, ciphertext).

Further features of system that simplify attack analysis:

5. Ciphertext is deterministic function of input $e$: i.e., inversion recovers all randomness used to create ciphertexts.
IND-CCA2 conversions

Classic McEliece follows best practices from literature:
1. Session key: feed random $e$ through standard hash function.
2. Ciphertext includes another hash of $e$ (“confirmation”).
3. Dec includes recomputation and verification of ciphertext.
4. KEM never fails: if inversion fails or ciphertext does not match, return hash of (secret, ciphertext).

Further features of system that simplify attack analysis:
5. Ciphertext is deterministic function of input $e$: i.e., inversion recovers all randomness used to create ciphertexts.
6. There are no inversion failures for legitimate ciphertexts.
Classic McEliece highlights

- Security asymptotics unchanged by 40 years of cryptanalysis.
- Short ciphertexts.
- Efficient and straightforward conversion of OW-CPA PKE into IND-CCA2 KEM.
- Constant-time software implementations.
- FPGA implementation of full cryptosystem.
- Open-source (public domain) implementations.
- No patents.

Classic McEliece highlights

https://classic.mceliece.org/