Classic McEliece: conservative code-based cryptography

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

[1]University of Illinois at Chicago, [2]Osaka University,
[3]Technische Universiteit Eindhoven, [4]Intel Corporation, [5]Fraunhofer SIT,
[6]Florida Atlantic University, [7]Radboud University, [8]Inria, [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.

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.

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 $s = He$?

1962 Prange: simple attack idea
guiding sizes in 1978 McEliece.

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^λ security against Prange's attack.

Here $c_0 \approx 0.7418860694$.

40 years and more than 30 analysis papers later

1962 Prange; 1981 Clark–Cain, crediting Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk; 1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg; 1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau; 1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne; 1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier; 2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg; 2009 Bernstein (**post-quantum**); 2009 Finiasz–Sendrier; 2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae; 2012 Becker–Joux–May–Meurer; 2013 Hamdaoui–Sendrier; 2015 May–Ozerov; 2016 Canto Torres–Sendrier; 2017 Kachigar–Tillich (**post-quantum**); 2017 Both–May; 2018 Both–May; 2018 Kirshanova (**post-quantum**).

40 years and more than 30 analysis papers later

1962 Prange; 1981 Clark–Cain, crediting Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk; 1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg; 1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau; 1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne; 1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier; 2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg; 2009 Bernstein (**post-quantum**); 2009 Finiasz–Sendrier; 2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae; 2012 Becker–Joux–May–Meurer; 2013 Hamdaoui–Sendrier; 2015 May–Ozerov; 2016 Canto Torres–Sendrier; 2017 Kachigar–Tillich (**post-quantum**); 2017 Both–May; 2018 Both–May; 2018 Kirshanova (**post-quantum**).

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

40 years and more than 30 analysis papers later

1962 Prange; 1981 Clark–Cain, crediting Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk; 1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg; 1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau; 1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne; 1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier; 2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg; 2009 Bernstein (**post-quantum**); 2009 Finiasz–Sendrier; 2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae; 2012 Becker–Joux–May–Meurer; 2013 Hamdaoui–Sendrier; 2015 May–Ozerov; 2016 Canto Torres–Sendrier; 2017 Kachigar–Tillich (**post-quantum**); 2017 Both–May; 2018 Both–May; 2018 Kirshanova (**post-quantum**).

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

Replacing λ with 2λ 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.

`mceliece6960119` parameter set (2008 Bernstein–Lange–Peters):
$q = 8192$, $n = 6960$, $w = 119$.

`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.

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 $(\text{secret}, \text{ciphertext})$.

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 $(\text{secret}, \text{ciphertext})$.

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 $(\text{secret}, \text{ciphertext})$.

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.