Classic McEliece: conservative code-based cryptography

D. J. Bernstein

classic.mceliece.org

Fundamental literature:
1962 Prange (attack)
+ many more attack papers.
1968 Berlekamp (decoder).
1970–1971 Goppa (codes).
1978 McEliece (cryptosystem).
1986 Niederreiter (dual)
+ many more optimizations.

Submission is joint work with:

Tung Chou, osaka-u.ac.jp
Tanja Lange, tue.nl*
Ingo von Maurich
Rafael Misoczki, intel.com
Ruben Niederhagen,
fraunhofer.de
Edoardo Persichetti, fau.edu
Christiane Peters
Peter Schwabe, ru.nl*
Nicolas Sendrier, inria.fr*
Jakub Szefer, yale.edu
Wen Wang, yale.edu

*: PQCRYPTO institutions.

`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.

`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.

`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.

Very fast in hardware:
a few million cycles at 231MHz using 129059 modules, 1126 RAM blocks on Altera Stratix V FPGA.

`mceliece6960119` parameter set:
226 bytes for ciphertext.

`mceliece8192128` parameter set:
240 bytes for ciphertext.

`mceliece6960119` parameter set:
226 bytes for ciphertext.

`mceliece8192128` parameter set:
240 bytes for ciphertext.

Software: 295932 cycles for enc,
355152 cycles for dec
(decoding, hashing, etc.).

`mceliece6960119` parameter set:
226 bytes for ciphertext.

`mceliece8192128` parameter set:
240 bytes for ciphertext.

Software: 295932 cycles for enc,
355152 cycles for dec
(decoding, hashing, etc.).

Again very fast in hardware:
17140 cycles for decoding.

`mceliece6960119` parameter set:
226 bytes for ciphertext.

`mceliece8192128` parameter set:
240 bytes for ciphertext.

Software: 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.

Encoding and decoding

1978 McEliece public key: matrix A over $\mathbf{F}_2$.

Ciphertext: vector $C = Ab + e$. Ab is "codeword"; e is random weight-w "error vector".

Original proposal for 2^{64} security: 1024×512 matrix; $w = 50$.

Public key is secretly generated with "binary Goppa code" structure that allows efficient decoding: $C \mapsto Ab, e$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $a_1, \ldots, a_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $a_1, \ldots, a_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of
the map $v \mapsto \sum_i v_i/(x - a_i)$
from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$.
Typical dimension $n - w \lg q$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $a_1, \ldots, a_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of
the map $v \mapsto \sum_i v_i/(x - a_i)$
from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$.
Typical dimension $n - w \lg q$.

McEliece uses random matrix A
whose image is this code.

One-wayness (OW-CPA)

Fundamental security question:
Given random public key A and ciphertext $Ab + e$ for random b, e, can attacker efficiently find b, e?

One-wayness (OW-CPA)

Fundamental security question:
Given random public key A and ciphertext $Ab + e$ for random b, e, can attacker efficiently find b, e?

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

One-wayness (OW-CPA)

Fundamental security question:
Given random public key A and ciphertext $Ab + e$ for random b, e, can attacker efficiently find b, e?

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 $c_0 \approx 0.7418860694$.

$\geq$25 subsequent publications analyzing one-wayness of system:

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 Finiasz–Sendrier.
2011 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Meurer.
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier.

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 $c_0 \approx 0.7418860694$.

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 (and is probably massive overkill), as in symmetric crypto.

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 (and is probably massive overkill), as in symmetric crypto.

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

`mceliece8192128` parameter set: $q = 8192$, $n = 8192$, $w = 128$.

McEliece's system prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security: e.g., Niederreiter's dual PKE; e.g., many decoding speedups. Classic McEliece uses all this.

McEliece's system prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security:
e.g., Niederreiter's dual PKE;
e.g., many decoding speedups.
Classic McEliece uses all this.

Classic McEliece does *not* use variants whose security has not been studied as thoroughly:
e.g., replacing binary Goppa codes with other families of codes;
e.g., lattice-based cryptography.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $n \times k$ matrix G with $\Gamma = G \cdot \mathbf{F}_2^k$.

McEliece public key: G times random $k \times k$ invertible matrix.

Niederreiter key compression

Generator matrix for code Γ
of length n and dimension k:
$n \times k$ matrix G with $\Gamma = G \cdot \mathbf{F}_2^k$.

McEliece public key: G times
random $k \times k$ invertible matrix.

Niederreiter instead reduces G
to the unique generator matrix
in “systematic form”: bottom k
rows are $k \times k$ identity matrix I_k.
Public key T is top $n - k$ rows.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $n \times k$ matrix G with $\Gamma = G \cdot \mathbf{F}_2^k$.

McEliece public key: G times random $k \times k$ invertible matrix.

Niederreiter instead reduces G to the unique generator matrix in “systematic form”: bottom k rows are $k \times k$ identity matrix I_k. Public key T is top $n - k$ rows.

Pr $\approx$29% that systematic form exists. Security loss: $<$2 bits.

Niederreiter ciphertext compression

Use Niederreiter key $A = \begin{pmatrix} T \\ I_k \end{pmatrix}$.

McEliece ciphertext: $Ab + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext compression

Use Niederreiter key $A = \begin{pmatrix} T \\ I_k \end{pmatrix}$.

McEliece ciphertext: $Ab + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter:
$He \in \mathbf{F}_2^{n-k}$ where $H = (I_{n-k}|T)$.

Niederreiter ciphertext compression

Use Niederreiter key $A = \begin{pmatrix} T \\ I_k \end{pmatrix}$.

McEliece ciphertext: $Ab + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter:
$He \in \mathbf{F}_2^{n-k}$ where $H = (I_{n-k}|T)$.

Given H and Niederreiter's He,
can attacker efficiently find e?

Niederreiter ciphertext compression

Use Niederreiter key $A = \begin{pmatrix} T \\ I_k \end{pmatrix}$.

McEliece ciphertext: $Ab + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter: $He \in \mathbf{F}_2^{n-k}$ where $H = (I_{n-k}|T)$.

Given H and Niederreiter's He, can attacker efficiently find e?

If so, attacker can efficiently find b, e given A and $Ab + e$: compute $H(Ab + e) = He$; find e; compute b from Ab.

Sampling via sorting

How to generate
random permutation of $\mathbf{F}_q$?
One answer (see, e.g., Knuth):
generate q random numbers,
sort them together with $\mathbf{F}_q$.

Sampling via sorting

How to generate
random permutation of $\mathbf{F}_q$?
One answer (see, e.g., Knuth):
generate q random numbers,
sort them together with $\mathbf{F}_q$.

How to generate
random weight-w vector $e \in \mathbf{F}_2^n$?
One answer:
generate n random numbers,
sort them together with
$(1, 1, \ldots, 1, 0, 0, \ldots, 0)$.

Sampling via sorting

How to generate
random permutation of $\mathbf{F}_q$?
One answer (see, e.g., Knuth):
generate q random numbers,
sort them together with $\mathbf{F}_q$.

How to generate
random weight-w vector $e \in \mathbf{F}_2^n$?
One answer:
generate n random numbers,
sort them together with
$(1, 1, \ldots, 1, 0, 0, \ldots, 0)$.

Divergence analysis $\Rightarrow$ use 32-bit
random numbers for typical n.

Similar computations are used in other NIST submissions.

Similar computations are used in other NIST submissions.

To avoid timing attacks, use constant-time sorting networks.

Similar computations are used in other NIST submissions.

To avoid timing attacks, use constant-time sorting networks.

NTRU Prime (Bernstein, Chuengsatiansup, Lange, van Vredendaal): new vectorized constant-time sorting software using Batcher's merge exchange.

Similar computations are used in other NIST submissions.

To avoid timing attacks, use constant-time sorting networks.

NTRU Prime (Bernstein, Chuengsatiansup, Lange, van Vredendaal): new vectorized constant-time sorting software using Batcher's merge exchange.

Optimized non-constant-time radix sort in Intel's Integrated Performance Primitives library is . . .

Similar computations are used in other NIST submissions.

To avoid timing attacks, use constant-time sorting networks.

NTRU Prime (Bernstein, Chuengsatiansup, Lange, van Vredendaal): new vectorized constant-time sorting software using Batcher's merge exchange.

Optimized non-constant-time radix sort in Intel's Integrated Performance Primitives library is . . . $5\times$ slower than this.

Much more on performance

See, e.g., the following papers and references cited therein:

2013 Bernstein–Chou–Schwabe "McBits: fast constant-time code-based cryptography".

2017 Chou "McBits revisited".

2017 Wang–Szefer–Niederhagen "FPGA-based key generator for the Niederreiter cryptosystem using binary Goppa codes".

2018 Wang–Szefer–Niederhagen, FPGA cryptosystem, to appear.

IND-CCA2 conversions

Classic McEliece aims for stronger security goal than original McEliece paper: indistinguishability vs. adaptive chosen-ciphertext attacks.

Many protocols need this.

IND-CCA2 conversions

Classic McEliece aims for stronger security goal than original McEliece paper: indistinguishability vs. adaptive chosen-ciphertext attacks. Many protocols need this.

Useful simplification: Encrypt user's plaintext with AES-GCM. Goal for public-key system: transmit random AES-GCM key. i.e. obtain IND-CCA2 PKE by designing IND-CCA2 KEM.

Want future auditors to be confident in long-term security. Classic McEliece follows best practices from literature:

1. Session key: feed random e through standard hash function.

Want future auditors to be confident in long-term security. 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").

Want future auditors to be confident in long-term security. 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.

Want future auditors to be confident in long-term security. 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.

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.

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.

Intuition for attackers:
can't predict session key without knowing e in advance;
can't generate fake ciphertexts;
dec doesn't reveal anything.

To *some* extent, intuition is captured by security proofs.

Attack of type T against KEM implies attack against P.

To *some* extent, intuition is captured by security proofs. Attack of type T against KEM implies attack against P.

Measuring quality of proofs:

- Security of P.
  Useless if P is weak;
  questionable if P is unstudied.

To *some* extent, intuition is captured by security proofs. Attack of type T against KEM implies attack against P.

Measuring quality of proofs:

- Security of P. Useless if P is weak; questionable if P is unstudied.
- Tightness of implication. Most proofs are not tight.

To *some* extent, intuition is captured by security proofs. Attack of type T against KEM implies attack against P.

Measuring quality of proofs:

- Security of P.
  Useless if P is weak;
  questionable if P is unstudied.

- Tightness of implication.
  Most proofs are not tight.

- Breadth of T.
  ROM? QROM? etc.

To *some* extent, intuition is captured by security proofs. Attack of type T against KEM implies attack against P.

Measuring quality of proofs:

- Security of P.
  Useless if P is weak; questionable if P is unstudied.

- Tightness of implication.
  Most proofs are not tight.

- Breadth of T.
  ROM? QROM? etc.

- Level of verification of proof.

Reasonable near-future goal:
formally verified tight proof
of IND-CCA2 security of KEM
against all ROM attacks
(maybe all QROM attacks)
assuming OW-CPA for McEliece.

Reasonable near-future goal:
formally verified tight proof
of IND-CCA2 security of KEM
against all ROM attacks
(maybe all QROM attacks)
assuming OW-CPA for McEliece.

2002 Dent (Theorem 8)
uses 1, 2, 3, 5, 6.
Proves tight IND-CCA2 security
against ROM attacks
under OW-CPA assumption.

Reasonable near-future goal:
formally verified tight proof
of IND-CCA2 security of KEM
against all ROM attacks
(maybe all QROM attacks)
assuming OW-CPA for McEliece.

2002 Dent (Theorem 8)
uses 1, 2, 3, 5, 6.
Proves tight IND-CCA2 security
against ROM attacks
under OW-CPA assumption.

2012 Persichetti (Theorem 5.1):
4 allows simpler proof strategy.

2017 Saito–Xagawa–Yamakawa ("XYZ" thm) uses 1, 3, 4, 5, 6. Proves tight IND-CCA2 security against QROM attacks under stronger assumptions.

Our KEM has 1, 2, 3, 4, 5, 6; all of these proof strategies appear to be applicable. See Classic McEliece submission.

Ongoing work to modularize, generalize, merge, verify proofs.

2017 Hofheinz–Hövelmanns–Kiltz: improved modularization.