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.

McEliece: conservative
sed cryptography

ernstein

c.mceliece.org

ental literature:
ange (attack)
more attack papers.
rlekamp (decoder).
71 Goppa (codes).
cEliece (cryptosystem).
ederreiter (dual)
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.

mcelie
1047319
13908 by

mcelie
1357824
14080 by

conservative
graphy

ce.org

ature:
ck)
ack papers.
decoder).
(codes).
yptosystem).
(dual)
imizations.

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.

mceliece696011
1047319 bytes for
13908 bytes for se

mceliece819212
1357824 bytes for
14080 bytes for se

've

·

n).

.

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` paramete

1047319 bytes for public key

13908 bytes for secret key.

`mceliece8192128` paramete

1357824 bytes for public key

14080 bytes for secret key.

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.

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.

Current software: billions of cycles to generate a key; not much optimization effort yet.

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.

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.

ion is joint work with:

iou, osaka-u.ac.jp

ange, tue.nl*

n Maurich

Misoczki, intel.com

Niederhagen,

unhofer.de

Persichetti, fau.edu

ne Peters

hwabe, ru.nl*

Sendrier, inria.fr*

zefer, yale.edu

ang, yale.edu

RYPTO 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.

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.

mcelie

226 byte

mcelie

240 byte

t work with:

a-u.ac.jp

.nl*

ntel.com

en,

de

ti, fau.edu

u.nl*

inria.fr*

e.edu

.edu

nstitutions.

`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.

`mceliece696011`
226 bytes for ciph

`mceliece819212`
240 bytes for ciph

h:

p

n

lu

*

.

`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` paramete
226 bytes for ciphertext.

`mceliece8192128` paramete
240 bytes for ciphertext.

`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:
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.

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

`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.

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

Again very fast in hardware:
17140 cycles for decoding.

`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.

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.

ce6960119 parameter set:
) bytes for public key.
ytes for secret key.

ce8192128 parameter set:
- bytes for public key.
ytes for secret key.

software: billions of cycles
ate a key; not much
ntion effort yet.

t in hardware:
illion cycles at 231MHz
9059 modules, 1126 RAM
n Altera Stratix V FPGA.

`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

1978 Mc
matrix A

Cipherte
Ab is "c
weight-w

Original
1024 × 5

Public k
with "bi
structure
decoding

9 parameter set:
public key.
cret key.

3 parameter set:
public key.
cret key.

billions of cycles
not much
t yet.

vare:
es at 231MHz
ules, 1126 RAM
Stratix V FPGA.

`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 deco

1978 McEliece pul
matrix A over $\mathbf{F}_2$.

Ciphertext: vector
Ab is "codeword";
weight-w "error ve

Original proposal f
1024×512 matrix

Public key is secre
with "binary Gopp
structure that allo
decoding: $C \mapsto A$

er set:

y.

er set:

y.

cycles

n

MHz

5 RAM

FPGA.

`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 $C = Ab$ -
Ab is "codeword"; e is rand
weight-w "error vector".

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

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

`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: $C \mapsto Ab, e$.

ce6960119 parameter set:
es for ciphertext.

ce8192128 parameter set:
es for ciphertext.

e: 295932 cycles for enc,
cycles for dec
ng, hashing, etc.).

ery fast in hardware:
ycles for decoding.

ak parameters for
aller ciphertexts,
h 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 G

Paramet

$w \in \{2,$

$n \in \{w\,$l

9 parameter set:
ertext.

3 parameter set:
ertext.

cycles for enc,
dec
g, etc.).

hardware:
ecoding.

eters for
rtexts,
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 cod

Parameters: $q \in \{$
$w \in \{2, 3, \ldots, \lfloor(q$
$n \in \{w \lg q + 1, \ldots$

er set:

er set:

enc,

e.

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

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

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

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

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

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 $n - w \lg q$.

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

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.

g and decoding

cEliece public key:
A over $\mathbf{F}_2$.

xt: vector $C = Ab + e$.
odeword"; e is random
v "error vector".

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

ey is secretly generated
nary Goppa code"
e that allows efficient
g: $C \mapsto Ab, e$.

Binary Goppa codes

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

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-way

Fundam
Given ra
ciphertex
can atta

oding

blic key:

$C = Ab + e$.

; e is random

ector”.

for 2^{64} security:

x; $w = 50$.

tly generated

a code”

ws efficient

b, e.

Binary Goppa codes

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

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

Fundamental secu
Given random pub
ciphertext $Ab + e$
can attacker effici

+ e.
om

curity:

ted

t

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 questi
Given random public key A a
ciphertext $Ab + e$ for rando
can attacker efficiently find

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?

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?

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

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?

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 $c_0 \approx 0.7418860694$.

Goppa codes

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

distinct $a_1, \ldots, a_n \in \mathbf{F}_q$;
reducible degree-w
ial $g \in \mathbf{F}_q[x]$.

ode: kernel of
$v \mapsto \sum_i v_i/(x - a_i)$
to $\mathbf{F}_q[x]/g$.
dimension $n - w \lg q$.

e uses random matrix A
mage 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?

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 sub
analyzin

1981 Cla
cre
1988 Le
1988 Le
1989 Kr
1989 Ste
1989 Du
1990 Co
1990 va
1991 Du
1991 Co
1993 Ch

es

$\{8, 16, 32, \ldots\}$;

$-1)/\lg q\rfloor\}$;

$., q-1, q\}$.

$_1, \ldots, a_n \in \mathbf{F}_q$;

degree-w

$_q[x]$.

el of

$v_i/(x-a_i)$

g.

$n - w \lg q$.

dom matrix A

is 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?

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 p

analyzing one-way

1981 Clark–Cain,

crediting Om

1988 Lee–Brickell.

1988 Leon.

1989 Krouk.

1989 Stern.

1989 Dumer.

1990 Coffey–Good

1990 van Tilburg.

1991 Dumer.

1991 Coffey–Good

1993 Chabanne–C

. . .};
q⌋};
q}.
$\in \mathbf{F}_q$;

)

q.

ix A

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 sys

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–Farr
1993 Chabanne–Courteau.

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 $c_0 \approx 0.7418860694$.

≥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.

yness (OW-CPA)

ental security question:
ndom public key A and
xt $Ab + e$ for random b, e,
cker efficiently find b, e?

ange: simple attack idea
sizes in 1978 McEliece.

Eliece system
ter key-size optimizations)
$+ o(1))\lambda^2(\lg \lambda)^2$-bit keys
∞ to achieve 2^λ security
Prange's attack.
≈ 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 Ch
1994 va
1994 Ca
1998 Ca
1998 Ca
2008 Be
2009 Be
va
2009 Fi
2011 Be
2011 M
2012 Be
2013 H
2015 M
2016 Ca

-CPA)

rity question:
lic key A and
for random b, e,
ently find b, e?

ple attack idea
78 McEliece.

em
e optimizations)
$^2(\lg \lambda)^2$-bit keys
eve 2^λ security
ttack.
360694.

≥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–Ch
1998 Canteaut–Ch
1998 Canteaut–Se
2008 Bernstein–La
2009 Bernstein–La
van Tilborg.
2009 Finiasz–Send
2011 Bernstein–La
2011 May–Meurer
2012 Becker–Joux
2013 Hamdaoui–S
2015 May–Ozerov
2016 Canto Torres

on:
and
m b, e,
b, e?

idea
ece.

tions)
t keys
curity

≥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–Peter
2009 Bernstein–Lange–Peter van Tilborg.
2009 Finiasz–Sendrier.
2011 Bernstein–Lange–Peter
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Me
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier

$\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.

bsequent publications
g one-wayness of system:

ark–Cain,
editing Omura.
e–Brickell.
on.
ouk.
ern.
umer.
offey–Goodman.
n Tilburg.
umer.
offey–Goodman–Farrell.
abanne–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 McE
uses (c_0
as $\lambda \rightarrow$
against a
Same c_0

ublications
ness of system:

nura.

lman.

lman–Farrell.
ourteau.

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 syst
uses $(c_0 + o(1))\lambda^2$
as $\lambda \to \infty$ to achi
against all attacks
Same $c_0 \approx 0.7418$

s

stem:

ell.

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 $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bi
as $\lambda \to \infty$ to achieve 2^λ sec
against all attacks known to
Same $c_0 \approx 0.7418860694$.

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 $c_0 \approx 0.7418860694$.

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 $c_0 \approx 0.7418860694$.

Replacing λ with 2λ stops all known *quantum* attacks (and is probably massive overkill), as in symmetric crypto.

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 $c_0 \approx 0.7418860694$.

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: $q = 8192$, $n = 8192$, $w = 128$.

abaud.
n Tilburg.
nteaut–Chabanne.
nteaut–Chabaud.
nteaut–Sendrier.
rnstein–Lange–Peters.
rnstein–Lange–Peters–
n Tilborg.
niasz–Sendrier.
rnstein–Lange–Peters.
ay–Meurer–Thomae.
cker–Joux–May–Meurer.
mdaoui–Sendrier.
ay–Ozerov.
nto 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 $c_0 \approx 0.7418860694$.

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
huge am

Some wo
while cle
e.g., Nie
e.g., ma
Classic M

habanne.
habaud.
endrier.
ange–Peters.
ange–Peters–

drier.
ange–Peters.
–Thomae.
–May–Meurer.
endrier.
.
s–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 $c_0 \approx 0.7418860694$.

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
huge amount of fo

Some work improv
while clearly prese
e.g., Niederreiter's
e.g., many decodin
Classic McEliece u

rs.

rs–

rs.

urer.

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
huge amount of followup wo

Some work improves efficien
while clearly preserving secu
e.g., Niederreiter's dual PKE
e.g., many decoding speedu
Classic McEliece uses all thi

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.

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.

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.

Eliece system
$+ o(1))\lambda^2(\lg \lambda)^2$-bit keys
∞ to achieve 2^λ security
all attacks known today.
≈ 0.7418860694.

g λ with 2λ
known *quantum* attacks
probably massive overkill),
mmetric crypto.

ce6960119 parameter set
ernstein–Lange–Peters):
2, $n = 6960$, $w = 119$.

ce8192128 parameter set:
2, $n = 8192$, $w = 128$.

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.

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.

Niederre

Generato
of length
$n \times k$ m

McEliece
random

em
$^2(\lg\lambda)^2$-bit keys
eve 2^λ security
known today.
860694.

2λ
uantum attacks
nassive overkill),
ypto.

9 parameter set
ange–Peters):
60, $w = 119$.

3 parameter set:
92, $w = 128$.

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.

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 co

Generator matrix f
of length n and di
$n \times k$ matrix G wi

McEliece public ke
random $k \times k$ inve

t keys
curity
oday.

tacks
erkill),

er set
ers):
19.

er set:
28.

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 compressio

Generator matrix for code Γ of length n and dimension k $n \times k$ matrix G with $\Gamma = G$

McEliece public key: G time random $k \times k$ invertible mat

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.

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 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.

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 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.

e's system prompted a
nount of followup work.

ork improves efficiency
early preserving security:
ederreiter's dual PKE;
ny decoding speedups.
McEliece uses all this.

McEliece does *not* use
whose security has not
udied as thoroughly:
lacing binary Goppa codes
er families of codes;
tice-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 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.

Niederre

Use Nied

McEliece

prompted a
ollowup work.

ves efficiency
rving security:
dual PKE;
ng speedups.
uses all this.

does *not* use
curity has not
oroughly:
ary Goppa codes
s of codes;
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 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 cipher

Use Niederreiter k

McEliece ciphertex

l a
ork.

cy
rity:
E;
ps.
s.

se
not

codes

phy.

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 ≈29% that systematic form exists. Security loss: <2 bits.

Niederreiter ciphertext comp

Use Niederreiter key $A = ($

McEliece ciphertext: $Ab + e$

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

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 ≈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, 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 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 ≈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, 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.

iter key compression

or matrix for code Γ
n n and dimension k:
atrix G with $\Gamma = G \cdot \mathbf{F}_2^k$.

e public key: G times
$k \times k$ invertible matrix.

iter instead reduces G
nique generator matrix
ematic form": bottom k
$k \times k$ identity matrix I_k.
ey T is top $n - k$ rows.

% that systematic form
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, 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.

Samplin

How to
random
One ans
generate
sort ther

ompression

for code Γ

mension k:

th $\Gamma = G \cdot \mathbf{F}_2^k$.

ey: G times

ertible matrix.

d reduces G

erator matrix

m": bottom k

entity matrix I_k.

p $n - k$ rows.

tematic form

ss: <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, 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 sorti

How to generate
random permutati
One answer (see, e
generate q random
sort them together

n

:

$\cdot\ \mathbf{F}_2^k$.

es

trix.

G

rix

m k

rix I_k.

ows.

orm

s.

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., Knut
generate q random numbers
sort them together with $\mathbf{F}_q$.

Niederreiter ciphertext compression

Use Niederreiter key $A = \left(\frac{T}{I_k}\right)$.

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 $\mathbf{F}_q$.

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 $\mathbf{F}_q$.

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

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 $\mathbf{F}_q$.

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.

iter ciphertext compression

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

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

iter ciphertext, shorter:
$^{n-k}$ where $H = (I_{n-k}|T)$.

and Niederreiter's He,
cker efficiently find e?

tacker can efficiently
given A and $Ab + e$:
e $H(Ab + e) = He$;
ompute 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 $\mathbf{F}_q$.

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 c
used in c

rtext compression

ey $A = \begin{pmatrix} T \\ I_k \end{pmatrix}$.

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

rtext, shorter:

$H = (I_{n-k}|T)$.

erreiter's He,

ently find e?

efficiently

nd $Ab + e$:

$e) = He$;

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 $\mathbf{F}_q$.

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 computatio

used in other NIST

pression

T
$\overline{T_k}\Big)$.

$e \in \mathbf{F}_2^n$.

ter:

$_k|T)$.

He,

e?

y

e:

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 submissi

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.

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.

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

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.

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.

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.

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 . . .

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.

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× slower than this.

g via sorting

generate
permutation of $\mathbf{F}_q$?
wer (see, e.g., Knuth):
e q random numbers,
m together with $\mathbf{F}_q$.

generate
weight-w vector $e \in \mathbf{F}_2^n$?
wer:
e n random numbers,
m together with
, $1, 0, 0, \ldots, 0$).

nce analysis ⇒ use 32-bit
numbers for typical n.

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× slower than this.

Much m

See, e.g.
and refe

2013 Be
"McBits
code-bas

2017 Ch

2017 Wa
"FPGA-
the Nied
using bi

2018 Wa
FPGA c

ng

on of $\mathbf{F}_q$?
e.g., Knuth):
n numbers,
r with $\mathbf{F}_q$.

vector $e \in \mathbf{F}_2^n$?

n numbers,
r with
. . , 0).

is $\Rightarrow$ use 32-bit
for typical n.

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× slower than this.

Much more on per

See, e.g., the follo
and references cite

2013 Bernstein–Ch
"McBits: fast con
code-based crypto

2017 Chou "McBi

2017 Wang–Szefer
"FPGA-based key
the Niederreiter cr
using binary Gopp

2018 Wang–Szefer
FPGA cryptosyste

h):

;,

$\in \mathbf{F}_2^n$?

,

32-bit

n.

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× slower than this.

Much more on performance

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

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

2017 Chou "McBits revisited

2017 Wang–Szefer–Niederha
"FPGA-based key generator
the Niederreiter cryptosyster
using binary Goppa codes".

2018 Wang–Szefer–Niederha
FPGA cryptosystem, to appe

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× 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.

computations are
other NIST submissions.

d timing attacks, use
t-time sorting networks.

Prime (Bernstein,
atiansup, Lange, van
aal): new vectorized
t-time sorting software
atcher's merge exchange.

ed non-constant-time
rt in Intel's Integrated
ance Primitives library
× 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-CC

Classic M
stronger
original
indisting
chosen-c
Many pr

ons are
T submissions.

ttacks, use
ing networks.

rnstein,
Lange, van
vectorized
ing software
erge exchange.

nstant-time
s Integrated
itives library
han 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 conver

Classic McEliece a
stronger security g
original McEliece p
indistinguishability
chosen-ciphertext
Many protocols ne

ons.

e

rks.

n

d

re

ange.

e

ed

ry

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. adapt chosen-ciphertext attacks. Many protocols need 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.

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.

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.

ore on performance

., the following papers
rences cited therein:

rnstein–Chou–Schwabe
: fast constant-time
sed cryptography".

ou "McBits revisited".

ang–Szefer–Niederhagen
based key generator for
lerreiter cryptosystem
nary Goppa codes".

ang–Szefer–Niederhagen,
ryptosystem, 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.

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 fu
confiden
Classic M
practices

1. Sessio
through

rformance

wing papers
ed therein:

hou–Schwabe
stant-time
graphy”.

ts revisited”.

r–Niederhagen
generator for
yptosystem
a codes”.

r–Niederhagen,
m, 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.

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 audit
confident in long-t
Classic McEliece f
practices from lite

1. Session key: fe
through standard

ers

abe
e

d".

agen
for
m

agen,
ear.

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 secur
Classic McEliece follows bes
practices from literature:

1. Session key: feed random
through standard hash funct

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.

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.

2. Ciphertext includes another hash of e ("confirmation").

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.

2. Ciphertext includes another hash of e ("confirmation").

3. Dec includes recomputation and verification of ciphertext.

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.

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).

A2 conversions

McEliece aims for
security goal than
McEliece paper:
guishability vs. adaptive
ciphertext attacks.
rotocols need this.

implification: Encrypt
aintext with AES-GCM.
public-key system:
t random AES-GCM key.
in IND-CCA2 PKE
ning 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.

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
that sim

5. Ciphe
function
inversion
used to

sions

ims for
goal than
paper:
vs. adaptive
attacks.
eed this.

on: Encrypt
th AES-GCM.
system:
AES-GCM key.
CA2 PKE
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.

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 o
that simplify attac

5. Ciphertext is de
function of input e
inversion recovers
used to create cip

tive

ot
CM.

key.

M.

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 deterministi
function of input e: i.e.,
inversion recovers all randon
used to create ciphertexts.

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.

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.

6. There are no inversion failures for legitimate ciphertexts.

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.

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.

ture auditors to be
t in long-term security.
McEliece follows best
s from literature:

on key: feed random e
standard hash function.

ertext includes another
e ("confirmation").

ncludes recomputation
fication of ciphertext.

never fails: if inversion
ciphertext does not match,
ash 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.

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*
capturec
Attack c
implies a

ors to be
term security.
ollows best
rature:

ed random *e*
hash function.

udes another
mation").

ecomputation
' ciphertext.

s: if inversion
does not match,
cret, 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.

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, in
captured by securi
Attack of type *T* a
implies attack aga

ity.

t

n *e*

tion.

her

ion

t.

sion

match,

rtext).

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 KE
implies attack against P.

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.

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.

Measuring quality of proofs:

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

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.

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.

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.

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.

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.

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.

features of system
plify attack analysis:

ertext is deterministic
of input e: i.e.,
n recovers all randomness
create ciphertexts.

e are no inversion failures
mate ciphertexts.

n for attackers:
edict session key
knowing e in advance;
nerate fake ciphertexts;
sn't reveal anything.

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.

Reasona
formally
of IND-C
against a
(maybe
assuming

f system
ck analysis:

eterministic
e: i.e.,
all randomness
hertexts.

nversion failures
ertexts.

kers:
on key
e in advance;
e ciphertexts;
anything.

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-fu
formally verified ti
of IND-CCA2 secu
against all ROM a
(maybe all QROM
assuming OW-CPA

:

c

nness

ilures

ce;

xts;

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 KE
against all ROM attacks
(maybe all QROM attacks)
assuming OW-CPA for McE

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.

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.

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

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.

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.

e extent, intuition is
d by security proofs.
of type T against KEM
attack against P.

ng quality of proofs:

ty of P.
s if P is weak;
onable if P is unstudied.

ness of implication.
proofs are not tight.

th of T.
? QROM? etc.

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.

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 Sa
("XYZ"
Proves t
against (
under st

Our KEM
all of the
appear t
Classic M

Ongoing
generaliz

2017 Ho
improved

ntuition is
ty proofs.
against KEM
inst P.

of proofs:

veak;
P is unstudied.

plication.
not tight.

etc.

tion 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.

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–Xagaw
(“XYZ” thm) uses
Proves tight IND-
against QROM at
under stronger ass

Our KEM has 1, 2
all of these proof s
appear to be appli
Classic McEliece s

Ongoing work to r
generalize, merge,

2017 Hofheinz–Hö
improved modular

EM

died.

oof.

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–Yamaka
("XYZ" thm) uses 1, 3, 4, 5
Proves tight IND-CCA2 secu
against QROM attacks
under stronger assumptions.

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

Ongoing work to modularize
generalize, merge, verify pro

2017 Hofheinz–Hövelmanns-
improved modularization.

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.