McTiny: McEliece for tiny network servers

Daniel J. Bernstein,
uic.edu, rub.de

Joint work with:
Tanja Lange, tue.nl

My main question in this talk:
Shouldn't NIST PQC simply standardize Classic McEliece, discard the other 25 proposals?

classic.mceliece.org

submission team (alphabetical):

- me;
- 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.

e for tiny network servers

. Bernstein,

u, rub.de

rk with:

ange, tue.nl

---

n question in this talk:

n't NIST PQC simply
dize Classic McEliece,
the other 25 proposals?

classic.mceliece.org

submission team (alphabetical):

- me;
- 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.

History

Fundam

1962 Pra

+ many

1968 Be

1970–19

1978 Mc

1986 Ni

+ many

2017: C

NIST: “

to gener

other se

Classic M

network servers

n,

.nl

in this talk:

PQC simply

sic McEliece,

25 proposals?

classic.mceliece.org

submission team (alphabetical):

- me;
- 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.

History

Fundamental litera

1962 Prange (atta

+ many more atta

1968 Berlekamp (

1970–1971 Goppa

1978 McEliece (cr

1986 Niederreiter

+ many more opti

2017: Classic McE

NIST: "the submit

to generate param

other security cate

Classic McEliece,

ervers

lk:

ply

ece,

osals?

classic.mceliece.org

submission team (alphabetical):

- me;
- 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`.

History

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

2017: Classic McEliece, rou

NIST: “the submitters may
to generate parameter sets f
other security categories.” =
Classic McEliece, round 2.

classic.mceliece.org

submission team (alphabetical):

- me;
- 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.

History

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.

2017: Classic McEliece, round 1.

NIST: "the submitters may wish to generate parameter sets for other security categories." ⇒ Classic McEliece, round 2.

c.mceliece.org

on team (alphabetical):

Chou, osaka-u.ac.jp;
Lange, tue.nl;
on Maurich;
Misoczki, intel.com;
Niederhagen,
hofer.de;
do Persichetti, fau.edu;
iane Peters;
Schwabe, ru.nl;
s Sendrier, inria.fr;
Szefer, yale.edu;
Wang, yale.edu.

History

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.

2017: Classic McEliece, round 1.

NIST: "the submitters may wish to generate parameter sets for other security categories." ⇒ Classic McEliece, round 2.

Encodin

1978 Mc
matrix A
Normally

ce.org

alphabetical):

ka-u.ac.jp;

e.nl;

h;

intel.com;

gen,

;

etti, fau.edu;

s;

ru.nl;

, inria.fr;

ale.edu;

e.edu.

History

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.

2017: Classic McEliece, round 1.

NIST: "the submitters may wish to generate parameter sets for other security categories." ⇒ Classic McEliece, round 2.

Encoding and deco

1978 McEliece pu

matrix A over $\mathbf{F}_2$.

Normally $s \mapsto As$

cal):

jp;

om;

edu;

r;

History

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.

2017: Classic McEliece, round 1.

NIST: "the submitters may wish to generate parameter sets for other security categories." ⇒ Classic McEliece, round 2.

Encoding and decoding

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

Normally $s \mapsto As$ is injective

History

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.

2017: Classic McEliece, round 1.

NIST: "the submitters may wish to generate parameter sets for other security categories." ⇒ Classic McEliece, round 2.

Encoding and decoding

1978 McEliece public key: matrix A over $\mathbf{F}_2$.
Normally $s \mapsto As$ is injective.

History

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.

2017: Classic McEliece, round 1.

NIST: "the submitters may wish to generate parameter sets for other security categories." $\Rightarrow$ Classic McEliece, round 2.

Encoding and decoding

1978 McEliece public key: matrix A over $\mathbf{F}_2$.
Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$.
Uses secret "codeword" As, weight-w "error vector" e.

History

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.

2017: Classic McEliece, round 1.

NIST: "the submitters may wish to generate parameter sets for other security categories." $\Rightarrow$ Classic McEliece, round 2.

Encoding and decoding

1978 McEliece public key: matrix A over $\mathbf{F}_2$.
Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$.
Uses secret "codeword" As, weight-w "error vector" e.

1978 parameters for 2^{64} security goal: 1024×512 matrix, $w = 50$.

History

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.

2017: Classic McEliece, round 1.

NIST: "the submitters may wish to generate parameter sets for other security categories." $\Rightarrow$ Classic McEliece, round 2.

Encoding and decoding

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

Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$. Uses secret "codeword" As, weight-w "error vector" e.

1978 parameters for 2^{64} security goal: 1024×512 matrix, $w = 50$.

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

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

lassic McEliece, round 1.

the submitters may wish
ate parameter sets for
curity categories.” ⇒
McEliece, round 2.

Encoding and decoding

1978 McEliece public key: matrix A over $\mathbf{F}_2$. Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$. Uses secret “codeword” As, weight-w “error vector” e.

1978 parameters for 2^{64} security goal: 1024×512 matrix, $w = 50$.

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

Binary G

Paramet
$w \in \{2,$
$n \in \{w$ l

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

Eliece, round 1.

tters may wish
eter sets for
egories.” ⇒
round 2.

Encoding and decoding

1978 McEliece public key: matrix A over $\mathbf{F}_2$. Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$. Uses secret “codeword” As, weight-w “error vector” e.

1978 parameters for 2^{64} security goal: 1024×512 matrix, $w = 50$.

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

Binary Goppa cod

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

.

m).

.

nd 1.

wish

for

⇒

Encoding and decoding

1978 McEliece public key:
matrix A over $\mathbf{F}_2$.
Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$.
Uses secret "codeword" As,
weight-w "error vector" e.

1978 parameters for 2^{64} security
goal: 1024×512 matrix, $w = 50$.

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

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 $\mathbf{F}_2$.
Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$.
Uses secret "codeword" As,
weight-w "error vector" e.

1978 parameters for 2^{64} security
goal: 1024×512 matrix, $w = 50$.

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

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 $\mathbf{F}_2$.
Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$.
Uses secret "codeword" As,
weight-w "error vector" e.

1978 parameters for 2^{64} security
goal: 1024×512 matrix, $w = 50$.

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

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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Encoding and decoding

1978 McEliece public key: matrix A over $\mathbf{F}_2$. Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$. Uses secret "codeword" As, weight-w "error vector" e.

1978 parameters for 2^{64} security goal: 1024×512 matrix, $w = 50$.

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

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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$; monic irreducible degree-w polynomial $g \in \mathbf{F}_q[x]$.

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

Encoding and decoding

1978 McEliece public key: matrix A over $\mathbf{F}_2$. Normally $s \mapsto As$ is injective.

Ciphertext: vector $C = As + e$. Uses secret "codeword" As, weight-w "error vector" e.

1978 parameters for 2^{64} security goal: 1024×512 matrix, $w = 50$.

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

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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$; monic irreducible degree-w polynomial $g \in \mathbf{F}_q[x]$.

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

McEliece uses random matrix A whose image is this code.

g and decoding

cEliece public key:
A over $\mathbf{F}_2$.
y $s \mapsto As$ is injective.

xt: vector $C = As + e$.
ret "codeword" As,
v "error vector" e.

rameters for 2^{64} security
24 $\times$ 512 matrix, $w = 50$.

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

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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

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

is injective.

$C = As + e$.

word" As,

ector" e.

or 2^{64} security

matrix, $w = 50$.

tly generated

a code"

ws efficient

s, 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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

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

McEliece uses random matrix A
whose image is this code.

One-wayness (OW

Fundamental secu
Given random pub
ciphertext $As + e$
can attacker efficie

e.

+ e.

urity

$= 50$.

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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

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

McEliece uses random matrix A
whose image is this code.

One-wayness (OW-Passive)

Fundamental security questi
Given random public key A
ciphertext $As + e$ for randor
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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

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

McEliece uses random matrix A
whose image is this code.

One-wayness (OW-Passive)

Fundamental security question:
Given random public key A and
ciphertext $As + e$ for random s, e,
can attacker efficiently find s, 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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

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

McEliece uses random matrix A
whose image is this code.

One-wayness (OW-Passive)

Fundamental security question:
Given random public key A and
ciphertext $As + e$ for random s, e,
can attacker efficiently find s, 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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w polynomial $g \in \mathbf{F}_q[x]$.

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

McEliece uses random matrix A whose image is this code.

One-wayness (OW-Passive)

Fundamental security question:
Given random public key A and ciphertext $As + e$ for random s, e, can attacker efficiently find s, 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 $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;

reducible degree-w

ial $g \in \mathbf{F}_q[x]$.

ode: kernel of

$v \mapsto \sum_i v_i/(x - \alpha_i)$

to $\mathbf{F}_q[x]/g$.

dimension $n - w \lg q$.

e uses random matrix A

mage is this code.

One-wayness (OW-Passive)

Fundamental security question:
Given random public key A and ciphertext $As + e$ for random s, e, can attacker efficiently find s, 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 sub

analyzin

1981 Cla

cre

1988 Le

1988 Le

1989 Kr

1989 St

1989 Du

1990 Co

1990 va

1991 Du

1991 Co

1993 Ch

es

{8, 16, 32, . . .};
$- 1)/ \lg q\rfloor\}$;
$. . , q - 1, q\}$.

$\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
degree-w
$_q[x]$.

el of
$v_i/(x - \alpha_i)$
g.
$n - w \lg q$.

dom matrix A
is code.

One-wayness (OW-Passive)

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

. . .};

$\rfloor$};

q}.

$\in \mathbf{F}_q$;

)

q.

x A

One-wayness (OW-Passive)

Fundamental security question:
Given random public key A and ciphertext $As + e$ for random s, e, can attacker efficiently find s, 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 publication
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-Passive)

Fundamental security question:
Given random public key A and ciphertext $As + e$ for random s, e, can attacker efficiently find s, 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.

yness (OW-Passive)

ental security question:
ndom public key A and
xt $As + e$ for random s, e,
cker efficiently find s, 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.

≥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 Ma
2012 Be
2013 Ha
2015 Ma
2016 Ca

/-Passive)

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

ple attack idea
78 McEliece.

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

$\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–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
n s, e,
s, 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): $q = 8192$, $n = 6960$, $w = 119$.

Also in submission: `8192128`, `6688128`, `460896`, `348864`.

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

Also in submission: `8192128`, `6688128`, `460896`, `348864`.

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

Also in submission: `8192128`, `6688128`, `460896`, `348864`.

McEliece's system

huge amount of fo

Some work improv

while clearly prese

e.g., Niederreiter's

e.g., many decodi

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

Also in submission: `8192128`, `6688128`, `460896`, `348864`.

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 speedup 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): $q = 8192$, $n = 6960$, $w = 119$.

Also in submission: `8192128`, `6688128`, `460896`, `348864`.

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

Also in submission: 8192128, 6688128, 460896, 348864.

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

submission: 8192128,

3, 460896, 348864.

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 lengt

$n \times k$ m

McEliece

random

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

2λ
uantum attacks
assive overkill),
ypto.

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

: 8192128,
, 348864.

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.

8,

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 $\Gamma = G$

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
rk.
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 $A = \Big($

McEliece ciphertext: $As + e$

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

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

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

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

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 s, e given A and $As + e$: compute $H(As + e) = He$; find e; compute s from As.

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

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 s, e given A and $As + e$:
compute $H(As + e) = He$;
find e; compute s from As.

The imm

Case stu
the most

2006 Sil
and CVP
studied f
both as
problems
pure and
physics a

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

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 s, e given A and $As + e$: compute $H(As + e) = He$; find e; compute s from As.

The immaturity of

Case study: SVP,
the most famous l

2006 Silverman: "
and CVP, have be
studied for more t
both as intrinsic m
problems and for a
pure and applied m
physics and crypto

n

r:

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

es

trix.

G

trix

m k

rix I_k.

ows.

orm

s.

Niederreiter ciphertext compression

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

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

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 s, e given A and $As + e$: compute $H(As + e) = He$; find e; compute s from As.

The immaturity of lattice at

Case study: SVP, the most famous lattice prol

2006 Silverman: "Lattices, S and CVP, have been intensiv studied for more than 100 ye both as intrinsic mathematic problems and for application pure and applied mathemati physics and cryptography."

Niederreiter ciphertext compression

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

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

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 s, e given A and $As + e$: compute $H(As + e) = He$; find e; compute s from As.

The immaturity of lattice attacks

Case study: SVP, the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Niederreiter ciphertext compression

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

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

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 s, e given A and $As + e$: compute $H(As + e) = He$; find e; compute s from As.

The immaturity of lattice attacks

Case study: SVP, the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

iter ciphertext compression

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

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

iter ciphertext, shorter:
$^{n-k}_2$ where $H = (I_{n-k}|T)$.

and Niederreiter's He,
cker efficiently find e?

tacker can efficiently
given A and $As + e$:
e $H(As + e) = He$;
ompute s from As.

The immaturity of lattice attacks

Case study: SVP,
the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SV
today: 2

Approx
believed
0.415: 2
0.415: 2

rtext compression

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

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

rtext, shorter:
$H = (I_{n-k}|T)$.

erreiter's He,
ently find e?

efficiently
nd $As + e$:
$e) = He$;
from As.

The immaturity of lattice attacks

Case study: SVP,
the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SVP algorith
today: $2^{\Theta(N)}$.

Approx c for some
believed to take ti
0.415: 2008 Nguy
0.415: 2010 Micci

pression

T / I_k).

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

ter:

$_k|T)$.

He,

e?

/

e:

The immaturity of lattice attacks

Case study: SVP,
the most famous lattice problem.

2006 Silverman: “Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography.”

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SVP algorithms known
today: $2^{\Theta(N)}$.

Approx c for some algorithm
believed to take time $2^{(c+o($

0.415: 2008 Nguyen–Vidick.

0.415: 2010 Micciancio–Vou

The immaturity of lattice attacks

Case study: SVP,
the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.

The immaturity of lattice attacks

Case study: SVP,
the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.

The immaturity of lattice attacks

Case study: SVP, the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:

0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.

The immaturity of lattice attacks

Case study: SVP,
the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.

The immaturity of lattice attacks

Case study: SVP,
the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

The immaturity of lattice attacks

Case study: SVP, the most famous lattice problem.

2006 Silverman: "Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography."

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-N lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:

0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

Lattice crypto: more attack avenues; even less understanding.

naturity of lattice attacks

udy: SVP,
t famous lattice problem.

verman: "Lattices, SVP
P, have been intensively
for more than 100 years,
intrinsic mathematical
s and for applications in
d applied mathematics,
and cryptography."

P algorithms known
: time $2^{\Theta(N \log N)}$ for
ll dimension-N lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:

0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

Lattice crypto: more attack avenues; even less understanding.

Agility, c

"You thi
That's c

f lattice attacks

attice problem.

‘Lattices, SVP
en intensively
han 100 years,
nathematical
applications in
mathematics,
ography.”

ms known
$(N \log N)$ for
on-N lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:

0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

Lattice crypto: more attack avenues; even less understanding.

Agility, diversity, e

“You think there c
That’s crazy! We

tacks

blem.

SVP
vely
ears,
cal
ns in
cs,

r
ces.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:

0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

Lattice crypto: more attack avenues; even less understanding.

Agility, diversity, etc.

“You think there can be onl
That’s crazy! We need back

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:

0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

Lattice crypto: more attack avenues; even less understanding.

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

Lattice crypto: more attack avenues; even less understanding.

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

McEliece has lower risk than lattice-based crypto. This doesn't mean that McEliece has zero risk.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:

0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

Lattice crypto: more attack avenues; even less understanding.

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

McEliece has lower risk than lattice-based crypto. This doesn't mean that McEliece has zero risk.

But there are also risks in standardizing more options: e.g., vulnerabilities are missed because cryptanalysts and implementors are spreading attention too thin.

Best SVP algorithms known today: $2^{\Theta(N)}$.

Approx c for some algorithms believed to take time $2^{(c+o(1))N}$:

0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.384: 2011 Wang–Liu–Tian–Bi.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
0.292: 2015 Becker–Ducas–Gama–Laarhoven.

Lattice crypto: more attack avenues; even less understanding.

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

McEliece has lower risk than lattice-based crypto. This doesn't mean that McEliece has zero risk.

But there are also risks in standardizing more options: e.g., vulnerabilities are missed because cryptanalysts and implementors are spreading attention too thin. OCB2 was published in 2004; standardized by ISO in 2009; complete break published in 2018.

P algorithms known
$2^{\Theta(N)}$.

c for some algorithms
to take time $2^{(c+o(1))N}$:
2008 Nguyen–Vidick.
2010 Micciancio–Voulgaris.
2011 Wang–Liu–Tian–Bi.
2013 Zhang–Pan–Hu.
2014 Laarhoven.
2015 Laarhoven–de Weger.
2015 Becker–Ducas–
Gama–Laarhoven.

crypto: more attack
even less understanding.

Agility, diversity, etc.

“You think there can be only one? That’s crazy! We need backups!”

McEliece has lower risk than lattice-based crypto. This doesn’t mean that McEliece has zero risk.

But there are also risks in standardizing more options: e.g., vulnerabilities are missed because cryptanalysts and implementors are spreading attention too thin. OCB2 was published in 2004; standardized by ISO in 2009; complete break published in 2018.

Integrity

“You wa
That’s c
post-qua

ms known

e algorithms
me $2^{(c+o(1))N}$:
en–Vidick.
ancio–Voulgaris.
g–Liu–Tian–Bi.
g–Pan–Hu.
hoven.
hoven–de Weger.
er–Ducas–
rhoven.

ore attack
understanding.

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

McEliece has lower risk than lattice-based crypto. This doesn't mean that McEliece has zero risk.

But there are also risks in standardizing more options: e.g., vulnerabilities are missed because cryptanalysts and implementors are spreading attention too thin. OCB2 was published in 2004; standardized by ISO in 2009; complete break published in 2018.

Integrity

"You want just en
That's crazy! Obv
post-quantum sigr

ns
1))N:
ulgaris.
n–Bi.
Weger.
nding.

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

McEliece has lower risk than lattice-based crypto. This doesn't mean that McEliece has zero risk.

But there are also risks in standardizing more options: e.g., vulnerabilities are missed because cryptanalysts and implementors are spreading attention too thin. OCB2 was published in 2004; standardized by ISO in 2009; complete break published in 2018.

Integrity

"You want just encryption? That's crazy! Obviously we post-quantum signatures too

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

McEliece has lower risk than lattice-based crypto. This doesn't mean that McEliece has zero risk.

But there are also risks in standardizing more options: e.g., vulnerabilities are missed because cryptanalysts and implementors are spreading attention too thin. OCB2 was published in 2004; standardized by ISO in 2009; complete break published in 2018.

Integrity

"You want just encryption? That's crazy! Obviously we need post-quantum signatures too!"

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

McEliece has lower risk than lattice-based crypto. This doesn't mean that McEliece has zero risk.

But there are also risks in standardizing more options: e.g., vulnerabilities are missed because cryptanalysts and implementors are spreading attention too thin. OCB2 was published in 2004; standardized by ISO in 2009; complete break published in 2018.

Integrity

"You want just encryption? That's crazy! Obviously we need post-quantum signatures too!"

Example: Google's NewHope experiment, modification of TLS.

- Server → client: E, one-time NewHope public key.
- Client → server: AES-GCM key **encrypted** to E.
- Server **signs** key exchange under its long-term RSA key.

Agility, diversity, etc.

"You think there can be only one? That's crazy! We need backups!"

McEliece has lower risk than lattice-based crypto. This doesn't mean that McEliece has zero risk.

But there are also risks in standardizing more options: e.g., vulnerabilities are missed because cryptanalysts and implementors are spreading attention too thin. OCB2 was published in 2004; standardized by ISO in 2009; complete break published in 2018.

Integrity

"You want just encryption? That's crazy! Obviously we need post-quantum signatures too!"

Example: Google's NewHope experiment, modification of TLS.

- Server $\rightarrow$ client: E, one-time NewHope public key.
- Client $\rightarrow$ server: AES-GCM key **encrypted** to E.
- Server **signs** key exchange under its long-term RSA key.

Must upgrade this protocol before attacker has quantum computer.

diversity, etc.

ink there can be only one?
razy! We need backups!"

e has lower risk than
ased crypto. This doesn't
at McEliece has zero risk.

re are also risks in
lizing more options: e.g.,
bilities are missed because
alysts and implementors
ading attention too thin.
was published in 2004;
lized by ISO in 2009;
e break published in 2018.

Integrity

"You want just encryption? That's crazy! Obviously we need post-quantum signatures too!"

Example: Google's NewHope experiment, modification of TLS.

- Server → client: E, one-time NewHope public key.
- Client → server: AES-GCM key **encrypted** to E.
- Server **signs** key exchange under its long-term RSA key.

Must upgrade this protocol before attacker has quantum computer.

More ge
Server si
server's
Client ve

tc.

can be only one?
need backups!"

r risk than
to. This doesn't
ce has zero risk.

risks in
e options: e.g.,
missed because
implementors
ntion too thin.
ed in 2004;
O in 2009;
blished in 2018.

Integrity

"You want just encryption? That's crazy! Obviously we need post-quantum signatures too!"

Example: Google's NewHope experiment, modification of TLS.

- Server → client: E, one-time NewHope public key.
- Client → server: AES-GCM key **encrypted** to E.
- Server **signs** key exchange under its long-term RSA key.

Must upgrade this protocol before attacker has quantum computer.

More general signa
Server signs messa
server's long-term
Client verifies sign

y one?
kups!"
n
oesn't
o risk.

e.g.,
cause
tors
thin.
4;
0;
2018.

Integrity

"You want just encryption? That's crazy! Obviously we need post-quantum signatures too!"

Example: Google's NewHope experiment, modification of TLS.

- Server → client: E, one-time NewHope public key.
- Client → server: AES-GCM key **encrypted** to E.
- Server **signs** key exchange under its long-term RSA key.

Must upgrade this protocol before attacker has quantum computer.

More general signature situa
Server signs message m und
server's long-term signature
Client verifies signature.

Integrity

"You want just encryption? That's crazy! Obviously we need post-quantum signatures too!"

Example: Google's NewHope experiment, modification of TLS.

- Server $\rightarrow$ client: E, one-time NewHope public key.
- Client $\rightarrow$ server: AES-GCM key **encrypted** to E.
- Server **signs** key exchange under its long-term RSA key.

Must upgrade this protocol before attacker has quantum computer.

More general signature situation: Server signs message m under server's long-term signature key. Client verifies signature.

Integrity

"You want just encryption? That's crazy! Obviously we need post-quantum signatures too!"

Example: Google's NewHope experiment, modification of TLS.

- Server $\rightarrow$ client: E, one-time NewHope public key.
- Client $\rightarrow$ server: AES-GCM key **encrypted** to E.
- Server **signs** key exchange under its long-term RSA key.

Must upgrade this protocol before attacker has quantum computer.

More general signature situation: Server signs message m under server's long-term signature key. Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\rightarrow$ server: AES-GCM key k encrypted to server's *long-term encryption key.*
- Server $\rightarrow$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

nt just encryption?
razy! Obviously we need
antum signatures too!"

e: Google's NewHope
ent, modification of TLS.
→ client: E,
me NewHope public key.
→ server:
GCM key **encrypted** to E.
signs key exchange
its long-term RSA key.

grade this protocol before
has quantum computer.

More general signature situation:
Server signs message m under
server's long-term signature key.
Client verifies signature.

Can protect integrity of m
without a signature system:

- Client → server:
  AES-GCM key k encrypted to
  server's *long-term encryption key.*
- Server → client:
  message m encrypted under k.

AES-GCM includes authentication
so client knows m is from server.

Advanta

Client kn

cryption?
viously we need
natures too!"

s NewHope
ication of TLS.
E,
ope public key.

encrypted to E.
exchange
erm RSA key.

protocol before
tum computer.

More general signature situation:
Server signs message m under server's long-term signature key.
Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\rightarrow$ server: AES-GCM key k encrypted to server's *long-term encryption key.*
- Server $\rightarrow$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

Advantages of this

Client knows m is

need
o!”

e
TLS.

key.

to E.
e
ey.

before
uter.

More general signature situation:
Server signs message m under server’s long-term signature key.
Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\rightarrow$ server: AES-GCM key k encrypted to server’s *long-term encryption key.*
- Server $\rightarrow$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

Advantages of this approach

Client knows m is fresh.

More general signature situation:
Server signs message m under server's long-term signature key.
Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\to$ server: AES-GCM key k encrypted to server's *long-term encryption key.*
- Server $\to$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

Advantages of this approach:

Client knows m is fresh.

More general signature situation:
Server signs message m under server's long-term signature key.
Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\rightarrow$ server: AES-GCM key k encrypted to server's *long-term encryption key.*
- Server $\rightarrow$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

More general signature situation:
Server signs message m under server's long-term signature key.
Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\rightarrow$ server: AES-GCM key k encrypted to server's *long-term encryption key.*
- Server $\rightarrow$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

Authenticates *and* encrypts.
Don't need 2nd encryption layer.

More general signature situation:
Server signs message m under server's long-term signature key.
Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\to$ server: AES-GCM key k encrypted to server's *long-term encryption key.*
- Server $\to$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

Authenticates *and* encrypts.
Don't need 2nd encryption layer.
— But "forward secrecy" needs an ephemeral encryption layer.

More general signature situation:
Server signs message m under server's long-term signature key.
Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\rightarrow$ server: AES-GCM key k encrypted to server's *long-term encryption key.*
- Server $\rightarrow$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

Authenticates *and* encrypts.
Don't need 2nd encryption layer.
— But "forward secrecy" needs an ephemeral encryption layer.

Advantage of signatures:
Signer can be offline.

More general signature situation:
Server signs message m under server's long-term signature key.
Client verifies signature.

Can protect integrity of m *without* a signature system:

- Client $\to$ server: AES-GCM key k encrypted to server's *long-term encryption key.*
- Server $\to$ client: message m encrypted under k.

AES-GCM includes authentication so client knows m is from server.

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

Authenticates *and* encrypts.
Don't need 2nd encryption layer.
— But "forward secrecy" needs an ephemeral encryption layer.

Advantage of signatures:
Signer can be offline.
— Designing for a disconnected future? Not relevant to TLS.

neral signature situation:
igns message m under
long-term signature key.
erifies signature.

tect integrity of m
a signature system:
$\rightarrow$ server:
GCM key k encrypted to
's *long-term encryption key.*
$\rightarrow$ client:
ge m encrypted under k.

M includes authentication
knows m is from server.

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

Authenticates *and* encrypts.
Don't need 2nd encryption layer.
— But "forward secrecy" needs an ephemeral encryption layer.

Advantage of signatures:
Signer can be offline.
— Designing for a disconnected future? Not relevant to TLS.

Time

Cycles o

params
348864
460896
668812[illegible]
696011[illegible]
819212[illegible]
348864
460896
668812[illegible]
696011[illegible]
819212[illegible]

ature situation:
age m under
signature key.
ature.

ity of m
re system:

encrypted to
m encryption key.

ypted under k.

s authentication
is from server.

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

Authenticates *and* encrypts.
Don't need 2nd encryption layer.
— But "forward secrecy" needs an ephemeral encryption layer.

Advantage of signatures:
Signer can be offline.
— Designing for a disconnected future? Not relevant to TLS.

Time

Cycles on Intel Ha

params	op	cy
348864	enc	45
460896	enc	82
6688128	enc	153
6960119	enc	154
8192128	enc	183
348864	dec	136
460896	dec	273
6688128	dec	320
6960119	dec	302
8192128	dec	324

tion:
er
key.

d to
ion key.

er k.

cation
erver.

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

Authenticates *and* encrypts.
Don't need 2nd encryption layer.
— But "forward secrecy" needs an ephemeral encryption layer.

Advantage of signatures:
Signer can be offline.
— Designing for a disconnected future? Not relevant to TLS.

Time

Cycles on Intel Haswell CPU

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

Advantages of this approach:

Client knows m is fresh.
— Already guaranteed for TLS, since m has client randomness.

Authenticates *and* encrypts.
Don't need 2nd encryption layer.
— But "forward secrecy" needs an ephemeral encryption layer.

Advantage of signatures:
Signer can be offline.
— Designing for a disconnected future? Not relevant to TLS.

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

ges of this approach:

nows *m* is fresh.

dy guaranteed for TLS,

has client randomness.

icates *and* encrypts.

eed 2nd encryption layer.

"forward secrecy" needs

meral encryption layer.

ge of signatures:

an be offline.

gning for a disconnected

Not relevant to TLS.

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

"Wait, y

most im

to have

params
348864
348864f
460896
460896f
668812
668812
696011
696011
819212
819212

s approach:

fresh.
teed for TLS,
randomness.

/ encrypts.
ncryption layer.
ecrecy" needs
ryption layer.

atures:
ne.
disconnected
ant to TLS.

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

"Wait, you're leav
most important co
to have such slow

params	op
348864	keyger
348864f	keyger
460896	keyger
460896f	keyger
6688128	keyger
6688128f	keyger
6960119	keyger
6960119f	keyger
8192128	keyger
8192128f	keyger

n:

TLS,
ess.

ayer.
eeds
er.

cted
S.

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

"Wait, you're leaving out th
most important cost! It's cr
to have such slow keygen!"

params	op	cy
348864	keygen	140870
348864f	keygen	82232
460896	keygen	441517
460896f	keygen	282869
6688128	keygen	1180468
6688128f	keygen	625470
6960119	keygen	1109340
6960119f	keygen	564570
8192128	keygen	933422
8192128f	keygen	678860

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

n Intel Haswell CPU core:

	op	cycles
	enc	45888
	enc	82684
3	enc	153372
9	enc	154972
3	enc	183892
	dec	136840
	dec	273872
3	dec	320428
9	dec	302460
3	dec	324008

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What
that this
a proble

swell CPU core:

ycles
5888
2684
3372
4972
3892
5840
3872
0428
2460
4008

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence
that this keygen ti
a problem for appl

J core:

“Wait, you’re leaving out the most important cost! It’s crazy to have such slow keygen!”

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence do we hav that this keygen time is a problem for applications?

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence do we have that this keygen time is a problem for applications?

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

ou're leaving out the
portant cost! It's crazy
such slow keygen!"

	op	cycles
	keygen	140870324
f	keygen	82232360
	keygen	441517292
f	keygen	282869316
3	keygen	1180468912
3f	keygen	625470504
9	keygen	1109340668
9f	keygen	564570384
3	keygen	933422948
3f	keygen	678860388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

Bytes co

params
348864
460896
6688128
6960119
8192128
348864
460896
6688128
6960119
8192128

"It's cra

ing out the
st! It's crazy
keygen!"

	cycles
n	140870324
n	82232360
n	441517292
n	282869316
n	1180468912
n	625470504
n	1109340668
n	564570384
n	933422948
n	678860388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

Bytes communicat

params	object
348864	ciphertex
460896	ciphertex
6688128	ciphertex
6960119	ciphertex
8192128	ciphertex
348864	key
460896	key
6688128	key
6960119	key
8192128	key

"It's crazy to have

e
azy

cles
0324
2360
7292
0316
3912
0504
0668
0384
2948
0388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

Bytes communicated

params	object	byte
348864	ciphertext	12
460896	ciphertext	18
6688128	ciphertext	24
6960119	ciphertext	22
8192128	ciphertext	24
348864	key	26112
460896	key	52416
6688128	key	104499
6960119	key	104731
8192128	key	135782

"It's crazy to have big keys!

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

t evidence do we have
s keygen time is
m for applications?

ic McEliece is designed
CCA2 security, so
n be generated once and
uge number of times.

iece's binary operations
well suited for hardware.
8 Wang–Szefer–
agen. Isn't this what's
portant for the future?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What ev
that the
a proble

do we have
me is
lications?

ce is designed
urity, so
rated once and
er of times.

ary operations
d for hardware.
zefer–
t this what's
r the future?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What evidence do
that these key size
a problem for appl

ve

ned

e and
s.

ons
ware.

t's
re?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What evidence do we have that these key sizes are a problem for applications?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What evidence do we have that these key sizes are a problem for applications?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

ommunicated

	object	bytes
	ciphertext	128
	ciphertext	188
3	ciphertext	240
9	ciphertext	226
3	ciphertext	240
	key	261120
	key	524160
3	key	1044992
9	key	1047319
3	key	1357824

zy to have big keys!"

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

2015 Mc
postqua
Use star
techniqu
etc.) to
commun

Each cip
the way
the serve
can ofte
much fa

Again IN

ted

	bytes
xt	128
xt	188
xt	240
xt	226
xt	240
	261120
	524160
	1044992
	1047319
	1357824

e big keys!"

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

2015 McGrew "Liv
postquantum cryp
Use standard netw
techniques (multic
etc.) to reduce cos
communicating pu

Each ciphertext ha
the way between t
the server, but pul
can often be retrie
much faster local

Again IND-CCA2

es

28

38

40

26

40

20

50

92

9

24

”

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

2015 McGrew "Living with postquantum cryptography":
Use standard networking techniques (multicasts, cach
etc.) to reduce cost of communicating public keys.

Each ciphertext has to trave
the way between the client a
the server, but public keys
can often be retrieved throu
much faster local network.

Again IND-CCA2 is critical.

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

vidence do we have
se key sizes are
m for applications?

e to, e.g., web-page size.

chive.org statistics:
web pages are >1.8MB.
web pages are >3.5MB.
web pages are >6.5MB.
s keep growing.

y browser receives one web
m multiple servers, but
ervers for more pages.
ze a big part of this?

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial o

Standard
strategy
of conne
up all m
for keepi

SYN flo

Server is
some co
connecti

we have
es are
lications?

web-page size.

g statistics:
are $>$1.8MB.
are $>$3.5MB.
are $>$6.5MB.
owing.

receives one web
e servers, but
more pages.
art of this?

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial of service

Standard low-cost
strategy: make a l
of connections to
up all memory ava
for keeping track o

SYN flood, HTTP

Server is forced to
some connections,
connections from l

size.

s:

MB.

MB.

MB.

ne web

but

s.

?

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial of service

Standard low-cost attack strategy: make a huge numb of connections to a server, f up all memory available on s for keeping track of connect

SYN flood, HTTP flood, etc

Server is forced to stop serv some connections, including connections from honest clie

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

cGrew "Living with ntum cryptography": ndard networking ues (multicasts, caching, reduce cost of nicating public keys.

phertext has to travel all between the client and er, but public keys n be retrieved through ster local network.

ND-CCA2 is critical.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny n** handles each inc without

ving with
tography":
vorking
casts, caching,
st of
ublic keys.

as to travel all
the client and
blic keys
eved through
network.

is critical.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network se**
handles and imme
each incoming net
without allocating

:

ing,

el all
and

gh

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network server**
handles and immediately for
each incoming network pack
without allocating any mem

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: “anonymous NFS”.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

f service

d low-cost attack
: make a huge number
ections to a server, filling
emory available on server
ing track of connections.

od, HTTP flood, etc.

s forced to stop serving
nnections, including
ons from honest clients.

ne Internet protocols
vulnerable to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: “anonymous NFS”.

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an “input continuation” fits into a network packet.

“Here’s
McEliece

attack
huge number
a server, filling
ilable on server
of connections.

flood, etc.

stop serving
including
honest clients.

protocols
to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural
McEliece can't po

per
illing
server
ions.
c.
ing
ents.
tack.

A **tiny network server**
handles and immediately forgets
each incoming network packet,
without allocating any memory.

Can use tiny network servers
to publish information.
Unauthenticated example from
last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario th
McEliece can't possibly hand

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: “anonymous NFS”.

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an “input continuation” fits into a network packet.

“Here’s a natural scenario that McEliece can’t possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: “anonymous NFS”.

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an “input continuation” fits into a network packet.

“Here’s a natural scenario that McEliece can’t possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It’s crazy if post-quantum standards can’t handle this!”

network server

and immediately forgets
oming network packet,
allocating any memory.

tiny network servers
sh information.
nticated example from
ury: "anonymous NFS".

ra–Nikander, 2005 Shieh–
irer modify any protocol
tiny network server
nput continuation"
a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstei
handles

erver

diately forgets
work packet,
any memory.

ork servers
tion.
example from
nymous NFS".

der, 2005 Shieh–
y any protocol
ork server
nuation"
packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstein–Lange "
handles this scena

gets
et,
ory.
s
om
IFS”.
Shieh–
tocol

“Here’s a natural scenario that McEliece can’t possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It’s crazy if post-quantum standards can’t handle this!”

Bernstein–Lange “McTiny” handles this scenario.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstein–Lange "McTiny" handles this scenario.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything.
Remaining problem:
within this session, encrypt to an ephemeral key for forward secrecy.

a natural scenario that
e can't possibly handle:
p memory floods,
a tiny network server.
rward secrecy,
the server to encrypt a
n key to an ephemeral
key sent by the client.
orces the public key
nto a network packet.
t 1500 bytes? Or 1280?
way, your key is too big.
y if post-quantum
ls can't handle this!"

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything. Remaining problem: within this session, encrypt to an ephemeral key for forward secrecy.

2. Clien
public ke

$$\begin{pmatrix} K_{1,1} \\ K_{2,1} \\ \vdots \\ K_{r,1} \end{pmatrix}$$

Each blo
to fit int

scenario that
ssibly handle:
floods,
twork server.
ecy,
r to encrypt a
n ephemeral
by the client.
public key
work packet.
es? Or 1280?
key is too big.
uantum
ndle this!"

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything.
Remaining problem:
within this session, encrypt to an ephemeral key for forward secrecy.

2. Client decompo
public key K into

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K \\ K_{2,1} & K_{2,2} & K \\ \vdots & \vdots & \\ K_{r,1} & K_{r,2} & K \end{pmatrix}$$

Each block is sma
to fit into a netwo

hat
dle:

er.

pt a
al
nt.

et.
80?
o big.

,

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything.

Remaining problem: within this session, encrypt to an ephemeral key for forward secrecy.

2. Client decomposes ephem
public key K into blocks: K

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots \\ \vdots & \vdots & \vdots & \ddots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots \end{pmatrix}$$

Each block is small enough to fit into a network packet.

Bernstein–Lange “McTiny” handles this scenario.

1. The easy part: Client encrypts session key to server’s long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server’s secret key can then decrypt everything.
Remaining problem:
within this session, encrypt to an ephemeral key for forward secrecy.

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \ldots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \ldots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \ldots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything.
Remaining problem:
within this session, encrypt to an ephemeral key for forward secrecy.

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

n–Lange “McTiny”
this scenario.

easy part: Client
. session key to server’s
m McEliece public key.
ablishes an encrypted
cated session.

r who records this session
r steals server’s secret key
n decrypt everything.
ng problem:
his session, encrypt to an
ral key for forward secrecy.

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client
containi
Server se

“McTiny”
rio.

Client
ey to server’s
e public key.
n encrypted
ion.

rds this session
rver’s secret key
everything.
m:
, encrypt to an
forward secrecy.

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \ldots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \ldots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \ldots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends on
containing several
Server sends back

er's
ey.
d

ession
et key
.

to an
ecrecy.

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \ldots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \ldots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \ldots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j$. Server sends back combinati

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j$. Server sends back combination.

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \ldots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \ldots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \ldots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j$. Server sends back combination.

5. Repeat to combine everything.

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j$. Server sends back combination.

5. Repeat to combine everything.

6. Server sends final Ke directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

2. Client decomposes ephemeral public key K into blocks: $K =$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j$. Server sends back combination.

5. Repeat to combine everything.

6. Server sends final Ke directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

Forward secrecy: Once cookie key and secret key for K are erased, client and server cannot decrypt.