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.

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.

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.

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, $w = 50$.

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

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

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 $n - w \lg q$.

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?

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.

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.

1993 Chabaud.

1994 van Tilburg.

1994 Canteaut–Chabanne.

1998 Canteaut–Chabaud.

1998 Canteaut–Sendrier.

2008 Bernstein–Lange–Peters.

2009 Bernstein–Lange–Peters–van Tilborg.

2009 Finiasz–Sendrier.

2011 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2012 Becker–Joux–May–Meurer.

2013 Hamdaoui–Sendrier.

2015 May–Ozerov.

2016 Canto Torres–Sendrier.

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

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

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

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

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

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

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.

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

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

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

Niederreiter key compression

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

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

Niederreiter key compression

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

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

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

Niederreiter key compression

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

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

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

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

Niederreiter ciphertext compression

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

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

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 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 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."

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.

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.

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.

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.

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.

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!"

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.

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.

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!"

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.

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.

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.

Advantages of this approach:

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

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.

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.

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.

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

"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?

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.

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!"

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

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.

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.

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.

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”.

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:

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

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

“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.

“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.

"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.

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.

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.

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.

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

5. Repeat to combine everything.

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.

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.