McTiny:
McEliece for tiny network servers

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

Tanja Lange, `tue.nl`

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

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

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

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

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

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

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

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

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

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

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 $\mathbf{F}_q[x]/g$.
Normally 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 $\mathbf{F}_q[x]/g$.
Normally dimension $n - w \lg q$.

McEliece uses random $G \in \mathbf{F}_2^{k \times n}$
whose image is this code.

One-wayness ("OW-Passive")

Fundamental security question: Can attacker efficiently find random m, e given random public key G and ciphertext $mG+e$?

One-wayness (“OW-Passive”)

Fundamental security question: Can attacker efficiently find random m, e given random public key G and ciphertext $mG+e$?

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

One-wayness ("OW-Passive")

Fundamental security question:
Can attacker efficiently find
random m, e given random public
key G and ciphertext $mG+e$?

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$26 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.

2017 Both–May.

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: 2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

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:
2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

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

NIST competition

2016: U.S. National Institute of Standards and Technology starts "post-quantum" competition.

2017: 69 complete submissions.

2019: NIST selects 26 submissions for round 2.

NIST competition

2016: U.S. National Institute of Standards and Technology starts "post-quantum" competition.

2017: 69 complete submissions.

2019: NIST selects
26 submissions for round 2.

"Classic McEliece": submission from team of 12 people.

Round-2 options:
8192128, 6960119, 6688128,
460896, 348864.

Is Classic McEliece same as 1978 McEliece? Not exactly.

1978 McEliece prompted a huge amount of followup work.

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

Classic McEliece also aims for more than OW-Passive security.

Niederreiter key compression

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

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k \times k}$.

Niederreiter key compression

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

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k\times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Niederreiter key compression

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

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k \times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k | R)$.

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

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

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

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

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

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

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

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$,
can attacker efficiently find e?

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

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

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$,
can attacker efficiently find e?

If so, attacker can efficiently
find m, e given G and $mG + e$:

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

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

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$,
can attacker efficiently find e?

If so, attacker can efficiently
find m, e given G and $mG + e$:
compute $H(mG + e)^\top = He^\top$;
find e; compute m from mG.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

No proof that changing codes preserves security level.

Classic McEliece: binary Goppa.

IND-CCA2 security

OW-Passive security is too weak.
Messages are not random.
Attackers choose ciphertexts
and observe reactions.

IND-CCA2 security

OW-Passive security is too weak.
Messages are not random.
Attackers choose ciphertexts
and observe reactions.

Classic McEliece does more work
for “IND-CCA2 security”.

Combines coding theory with
AES-GCM “authenticated cipher”
and SHA-3 “hash function”.

All messages are safe.
Reusing keys is safe.

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 $K = R^\top$ into blocks:

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

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

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

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

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

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j^\top$ 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 $K_{i,j}e_j^\top$. Server sends back combination.

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

5. Repeat to combine everything, including I_{n-k} part of H.

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

5. Repeat to combine everything, including I_{n-k} part of H.

6. Server sends final $He^\top$ directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

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

5. Repeat to combine everything, including I_{n-k} part of H.

6. Server sends final $He^\top$ directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

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

Classic McEliece recap

Security asymptotics unchanged by 40 years of cryptanalysis.

Ciphertexts among the shortest.

IND-CCA2 security.

Open-source implementations: fast constant-time software, also FPGA implementation.

No patents.

Big keys, but still compatible with tiny network servers.

`https://classic.mceliece.org`