Advanced

code-based cryptography

Daniel J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven
Lattice-basis reduction

Define $L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z} = \{(b, 24a + 17b) : a, b \in \mathbb{Z}\}$.

What is the shortest nonzero vector in $L$?
Lattice-basis reduction

Define $L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z}$

$= \{(b, 24a + 17b) : a, b \in \mathbb{Z}\}$.

What is the shortest nonzero vector in $L$?

$L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z}$
Lattice-basis reduction

Define $L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbb{Z}\}$.

What is the shortest nonzero vector in $L$?

$L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z}$
$= (-1, 7)\mathbb{Z} + (1, 17)\mathbb{Z}$
Lattice-basis reduction

Define \( L = (0, 24) \mathbb{Z} + (1, 17) \mathbb{Z} \)
\[ = \{(b, 24a + 17b) : a, b \in \mathbb{Z}\}. \]

What is the shortest nonzero vector in \( L \)?

\[
L = (0, 24) \mathbb{Z} + (1, 17) \mathbb{Z} \\
= (-1, 7) \mathbb{Z} + (1, 17) \mathbb{Z} \\
= (-1, 7) \mathbb{Z} + (3, 3) \mathbb{Z}
\]
Define $L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z}$

$= \{(b, 24a + 17b) : a, b \in \mathbb{Z}\}$.

What is the shortest nonzero vector in $L$?

$L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z}$

$= (-1, 7)\mathbb{Z} + (1, 17)\mathbb{Z}$

$= (-1, 7)\mathbb{Z} + (3, 3)\mathbb{Z}$

$= (-4, 4)\mathbb{Z} + (3, 3)\mathbb{Z}$.
Lattice-basis reduction

Define \( L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z} \)
\[
= \{(b, 24a + 17b) : a, b \in \mathbb{Z}\}.
\]

What is the shortest nonzero vector in \( L \)?

\[
L = (0, 24)\mathbb{Z} + (1, 17)\mathbb{Z}
= (-1, 7)\mathbb{Z} + (1, 17)\mathbb{Z}
= (-1, 7)\mathbb{Z} + (3, 3)\mathbb{Z}
= (-4, 4)\mathbb{Z} + (3, 3)\mathbb{Z}.
\]

\((-4, 4), (3, 3)\) are orthogonal.

Shortest vectors in \( L \) are
\( (0, 0), (3, 3), (-3, -3) \).
Another example:
Define $L = (0, 25)\mathbb{Z} + (1, 17)\mathbb{Z}$.

What is the shortest nonzero vector in $L$?
Another example:
Define \( L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z} \).

What is the shortest nonzero vector in \( L \)?

\( L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z} \)
Another example:
Define $L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$.

What is the shortest nonzero vector in $L$?

$L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$

$= (-1, 8)\mathbf{Z} + (1, 17)\mathbf{Z}$
Another example:
Define $L = (0, 25)\mathbb{Z} + (1, 17)\mathbb{Z}$.

What is the shortest nonzero vector in $L$?

$L = (0, 25)\mathbb{Z} + (1, 17)\mathbb{Z} = (-1, 8)\mathbb{Z} + (1, 17)\mathbb{Z} = (-1, 8)\mathbb{Z} + (3, 1)\mathbb{Z}$. 
Another example:
Define \( L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z} \).

What is the shortest nonzero vector in \( L \)?

\[
L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z} \\
= (-1, 8)\mathbf{Z} + (1, 17)\mathbf{Z} \\
= (-1, 8)\mathbf{Z} + (3, 1)\mathbf{Z}.
\]

Nearly orthogonal.
Shortest vectors in \( L \) are \((0, 0), (3, 1), (-3, -1)\).
Define $P = \mathbb{F}_2[x]$, 

$r_0 = (101000)_x = x^5 + x^3 \in P,$

$r_1 = (10011)_x = x^4 + x + 1 \in P,$

$L = (0, r_0)P + (1, r_1)P.$

What is the shortest nonzero vector in $L$?
Define $P = \mathbb{F}_2[x]$, 
$r_0 = (101000)_x = x^5 + x^3 \in P$, 
$r_1 = (10011)_x = x^4 + x + 1 \in P$, 
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in $L$?

$L = (0, 101000)P + (1, 10011)P$
Polynomial lattices

Define \( P = \mathbb{F}_2[x] \),

\[
\begin{align*}
    r_0 &= (101000)_x = x^5 + x^3 \in P, \\
    r_1 &= (10011)_x = x^4 + x + 1 \in P,
\end{align*}
\]

\( L = (0, r_0)P + (1, r_1)P \).

What is the shortest nonzero vector in \( L \)?

\[
\begin{align*}
    L &= (0, 101000)P + (1, 10011)P \\
        &= (10, 1110)P + (1, 10011)P
\end{align*}
\]
Define $P = \mathbb{F}_2[x]$,

$r_0 = (101000)x = x^5 + x^3 \in P,$

$r_1 = (10011)x = x^4 + x + 1 \in P,$

$L = (0, r_0)P + (1, r_1)P.$

What is the shortest nonzero vector in $L$?

$L = (0, 101000)P + (1, 10011)P$

$= (10, 1110)P + (1, 10011)P$

$= (10, 1110)P + (111, 1)P.$
Polynomial lattices

Define $P = \mathbb{F}_2[x]$, 
$r_0 = (101000)_x = x^5 + x^3 \in P$, 
$r_1 = (10011)_x = x^4 + x + 1 \in P$, 
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in $L$?

$L = (0, 101000)P + (1, 10011)P$
$= (10, 1110)P + (1, 10011)P$
$= (10, 1110)P + (111, 1)P$.

$(111, 1)$: shortest nonzero vector. 
$(10, 1110)$: shortest independent vector.
Degree of \((q, r) \in \mathbb{F}_2[x] \times \mathbb{F}_2[x]\) is defined as \(\max\{\deg q, \deg r\}\).
Degree of \((q, r) \in \mathbb{F}_2[x] \times \mathbb{F}_2[x]\)
is defined as \(\max\{\deg q, \deg r\}\).

Can use other metrics, or equivalently rescale \(L\).

e.g. Define \(L \subseteq \mathbb{F}_2[\sqrt{x}] \times \mathbb{F}_2[\sqrt{x}]\)as \((0, r_0 \sqrt{x})P + (1, r_1 \sqrt{x})P\).
Degree of \((q, r) \in F_2[x] \times F_2[x]\) is defined as \(\max\{\deg q, \deg r\}\).

Can use other metrics, or equivalently rescale \(L\).

e.g. Define \(L \subseteq F_2[\sqrt{x}] \times F_2[\sqrt{x}]\) as \((0, r_0 \sqrt{x})P + (1, r_1 \sqrt{x})P\).

Successive generators for \(L\):
\((0, 101000\sqrt{x})\), degree 5.5.
\((1, 10011\sqrt{x})\), degree 4.5.
\((10, 1110\sqrt{x})\), degree 3.5.
\((111, 1\sqrt{x})\), degree 2.
Warning: Sometimes shortest independent vector is after shortest nonzero vector.
Warning: Sometimes shortest independent vector is after shortest nonzero vector.

e.g. Define

\( r_0 = 101000, \ r_1 = 10111, \)

\( L = (0, r_0 \sqrt{x}) P + (1, r_1 \sqrt{x}) P. \)

Successive generators for \( L \):

\( (0, 101000 \sqrt{x}), \) degree 5.5.

\( (1, 10111 \sqrt{x}), \) degree 4.5.

\( (10, 110 \sqrt{x}), \) degree 2.5.

\( (1101, 11 \sqrt{x}), \) degree 3.
For any field $k$, any $r_0, r_1$ in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \mod r_1$, $r_3 = r_1 \mod r_2$, etc.
For any field $k$, any $r_0, r_1$ in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \mod r_1$, $r_3 = r_1 \mod r_2$, etc.

Extended: $q_0 = 0; q_1 = 1; q_{i+2} = q_i - \lfloor \frac{r_i}{r_{i+1}} \rfloor q_{i+1}$. Then $q_i r_1 \equiv r_i \pmod{r_0}$. 
For any field $k$, any $r_0, r_1$ in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \mod r_1$,
$r_3 = r_1 \mod r_2$, etc.

Extended: $q_0 = 0; q_1 = 1;
q_{i+2} = q_i - \lfloor r_i/r_{i+1} \rfloor q_{i+1}$.
Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have
$(0, r_0 \sqrt{x})P + (1, r_1 \sqrt{x})P =
(q_i, r_i \sqrt{x})P + (q_{i+1}, r_{i+1} \sqrt{x})P.$
For any field $k$, any $r_0, r_1$ in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \mod r_1$, $r_3 = r_1 \mod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$; $q_{i+2} = q_i - \left\lfloor r_i/r_{i+1} \right\rfloor q_{i+1}$. Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have
$$(0, r_0 \sqrt{x})P + (1, r_1 \sqrt{x})P = (q_i, r_i \sqrt{x})P + (q_{i+1}, r_{i+1} \sqrt{x})P.$$ Can continue until $r_{i+1} = 0$.

$\gcd\{r_0, r_1\} = r_i/\text{leadcoeff } r_i$. 
Reducing lattice basis for $L$ is a “half gcd” computation, stopping halfway to the gcd.
Reducing lattice basis for $L$ is a “half gcd” computation, stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases; $\deg q_{i+1} + \deg r_i = \deg r_0$. 
Reducing lattice basis for $L$ is a “half gcd” computation, stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases; $\deg q_{i+1} + \deg r_i = \deg r_0$.

Say $j$ is minimal with $\deg r_j \sqrt{x} \leq (\deg r_0)/2$.

Then $\deg q_j \leq (\deg r_0)/2$ so $\deg(q_j, r_j \sqrt{x}) \leq (\deg r_0)/2$.

Shortest nonzero vector.
Reducing lattice basis for $L$ is a “half gcd” computation, stopping halfway to the gcd.

deg $r_i$ decreases; deg $q_i$ increases; 
$\deg q_{i+1} + \deg r_i = \deg r_0$.

Say $j$ is minimal with 
$\deg r_j \sqrt{x} \leq (\deg r_0)/2$.
Then $\deg q_j \leq (\deg r_0)/2$ so 
$\deg(q_j, r_j \sqrt{x}) \leq (\deg r_0)/2$.

Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon} \sqrt{x})$ has degree 
$\deg r_0 \sqrt{x} - \deg(q_j, r_j \sqrt{x})$
for some $\epsilon \in \{-1, 1\}$.

Shortest independent vector.
Proof of “shortest”:
Take any \((q, r\sqrt{x})\) in lattice.
Proof of "shortest":
Take any \((q, r\sqrt{x})\) in lattice.

\[(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})\]

for some \(u, v \in P\).
Proof of “shortest”:
Take any \((q, r \sqrt{x})\) in lattice.

\[
(q, r \sqrt{x}) = u(q_j, r_j \sqrt{x}) \\
+ v(q_{j+\epsilon}, r_{j+\epsilon} \sqrt{x})
\]
for some \(u, v \in P\).

\[
q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0
\]
do \(v = \pm (rq_j - qr_j)/r_0\)
and \(u = \pm (qr_{j+\epsilon} - rq_{j+\epsilon})/r_0\).
Proof of “shortest”:
Take any \((q, r\sqrt{x})\) in lattice.

\[
(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})
\]
for some \(u, v \in P\).

\[
q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0
\]
so \(v = \pm (rq_j - qr_j)/r_0\)
and \(u = \pm (qr_{j+\epsilon} - rq_{j+\epsilon})/r_0\).

If \(\deg(q, r\sqrt{x})<\deg(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})\)
then \(\deg v < 0\) so \(v = 0\);
i.e., any vector in lattice shorter than \((q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})\)
is a multiple of \((q_j, r_j\sqrt{x})\).
Classical binary Goppa codes

Fix integer $n \geq 0$; integer $m \geq 1$ with $2^m \geq n$; integer $t \geq 0$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$; monic $g \in \mathbf{F}_{2^m}[x]$ of degree $t$ with $g(a_1) \cdots g(a_n) \neq 0$. 
Classical binary Goppa codes

Fix integer \( n \geq 0 \);
integer \( m \geq 1 \) with \( 2^m \geq n \);
integer \( t \geq 0 \);
distinct \( a_1, \ldots, a_n \in \mathbb{F}_{2^m} \);
monic \( g \in \mathbb{F}_{2^m}[x] \) of degree \( t \)
with \( g(a_1) \cdots g(a_n) \neq 0 \).

Note that \( x - a_i \)
has a reciprocal in \( \mathbb{F}_{2^m}[x]/g \).
Classical binary Goppa codes

Fix integer $n \geq 0$; integer $m \geq 1$ with $2^m \geq n$; integer $t \geq 0$; distinct $a_1, \ldots, a_n \in \mathbb{F}_{2^m}$; monic $g \in \mathbb{F}_{2^m}[x]$ of degree $t$ with $g(a_1) \cdots g(a_n) \neq 0$.

Note that $x - a_i$ has a reciprocal in $\mathbb{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbb{F}_2^n$ as set of $(c_1, \ldots, c_n)$ with $\sum_i c_i/(x - a_i) = 0$ in $\mathbb{F}_{2^m}[x]/g$.

Then $\#\Gamma \geq 2^{n-mt}$. 
Goal: Find $c \in \Gamma$ given

$v = c + e$, assuming $|e| \leq \frac{t}{2}$. 
Goal: Find $c \in \Gamma$ given $v = c + e$, assuming $|e| \leq t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$ to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.

Find shortest nonzero $(q_j, r_j\sqrt{x})$ in the lattice $L = (0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$. 
Goal: Find $c \in \Gamma$ given $v = c + e$, assuming $|e| \leq t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbb{F}_{2^m}[x]/g$ to $s \in \mathbb{F}_{2^m}[x]$ with $\deg s < t$.

Find shortest nonzero $(q_j, r_j \sqrt{x})$ in the lattice $L = (0, g \sqrt{x})\mathbb{F}_{2^m}[x] + (1, s \sqrt{x})\mathbb{F}_{2^m}[x]$.

Define $E, F \in \mathbb{F}_{2^m}[x]$ by $F = \prod_{i:e_i \neq 0} (x - a_i)$ and $E = \sum_i F e_i/(x - a_i)$.

Fact: $E/F = r_j/q_j$ so $F$ is monic denominator of $r_j/q_j$. 
Goal: Find $c \in \Gamma$ given $v = c + e$, assuming $|e| \leq t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $F_{2^m}[x]/g$ to $s \in F_{2^m}[x]$ with $\deg s < t$.

Find shortest nonzero $(q_j, r_j \sqrt{x})$ in the lattice $L = (0, g \sqrt{x})F_{2^m}[x] + (1, s \sqrt{x})F_{2^m}[x]$.

Define $E, F \in F_{2^m}[x]$ by $F = \prod_{i: e_i \neq 0} (x - a_i)$ and $E = \sum_i F e_i/(x - a_i)$.

Fact: $E/F = r_j/q_j$ so $F$ is monic denominator of $r_j/q_j$.

$e_i = 0$ if $F(a_i) \neq 0$.

$e_i = E(a_i)/F'(a_i)$ if $F(a_i) = 0$. 
This decoder “corrects $\lfloor t/2 \rfloor$ errors for $\Gamma$”.

Why does this work?

$$\sum_i e_i / (x - a_i) = E/F \quad \text{and} \quad \sum_i c_i / (x - a_i) = 0 \quad \text{in} \quad F_{2^m}[x]/g$$

so $s = E/F$ in $F_{2^m}[x]/g$

so $(F, E \sqrt{x}) \in L$. 
This decoder “corrects \( \lfloor t/2 \rfloor \) errors for \( \Gamma \).”

Why does this work?

\[
\sum_i e_i/(x - a_i) = E/F \text{ and } \sum_i c_i/(x - a_i) = 0 \text{ in } F_{2^m}[x]/g
\]

so \( s = E/F \) in \( F_{2^m}[x]/g \)

so \( (F, E \sqrt{x}) \in L \).

\( (F, E \sqrt{x}) \) is a short vector:
\[
\deg(F, E \sqrt{x}) \leq |e| \leq t/2
\]
\[
< t + 1/2 - \deg(q_j, r_j \sqrt{x}).
\]
This decoder “corrects ⌊t/2⌋ errors for Γ”.

Why does this work?

\[
\sum_i e_i/(x - a_i) = E/F \quad \text{and} \\
\sum_i c_i/(x - a_i) = 0 \quad \text{in } F_{2^m}[x]/g
\]

so \( s = E/F \) in \( F_{2^m}[x]/g \)

so \((F, E \sqrt{x}) \in L\).

\((F, E \sqrt{x})\) is a short vector:
\[
\deg(F, E \sqrt{x}) \leq |e| \leq t/2
\]

\(< t + 1/2 - \deg(q_j, r_j \sqrt{x})\).

Recall proof of “shortest”:
\( (F, E \sqrt{x}) \in (q_j, r_j \sqrt{x})F_{2^m}[x], \)
so \( E/F = r_j/q_j \). Done!
The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:

$\sum_i c_i/(x - a_i) = 0$ in $F_{2^m}[x]/g$ if

$\sum_i c_i/(x - a_i) = 0$ in $F_{2^m}[x]/g^2$. 
The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:

$\sum_i c_i/(x - a_i) = 0$ in $F_{2^m}[x]/g$ if $\sum_i c_i/(x - a_i) = 0$ in $F_{2^m}[x]/g^2$.

Amazing fact:

$\Gamma(g) = \Gamma(g^2)$ if $g$ is squarefree.
The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:

$\sum_i c_i/(x - a_i) = 0$ in $\mathbb{F}_{2^m}[x]/g$ if $\sum_i c_i/(x - a_i) = 0$ in $\mathbb{F}_{2^m}[x]/g^2$.

Amazing fact:

$\Gamma(g) = \Gamma(g^2)$ if $g$ is squarefree.

Previous decoder for $g^2$ corrects $t$ errors for $\Gamma(g^2)$, hence corrects $t$ errors for $\Gamma(g)$. 
The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:

$$\sum_i c_i/(x - a_i) = 0 \text{ in } \mathbb{F}_{2^m}[x]/g \text{ if }$$

$$\sum_i c_i/(x - a_i) = 0 \text{ in } \mathbb{F}_{2^m}[x]/g^2.$$ 

Amazing fact:

$\Gamma(g) = \Gamma(g^2)$ if $g$ is squarefree.

Previous decoder for $g^2$ corrects $t$ errors for $\Gamma(g^2)$, hence corrects $t$ errors for $\Gamma(g)$.

(Not covered in this talk: correcting $\approx t + t^2/n$ errors. See, e.g., “jet list decoding”.)
Proof: Assume

$$\sum_i c_i/(x - a_i) = 0 \text{ in } \mathbb{F}_{2^m}[x]/g.$$
Proof: Assume 
\[ \sum_i c_i/(x - a_i) = 0 \text{ in } \mathbf{F}_{2^m}[x]/g. \]

Write \( F = \prod_{i:c_i \neq 0} (x - a_i). \)

Then \( F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i) \)
so \( F'/F = \sum c_i/(x - a_i) \)
so \( F'/F = 0 \text{ in } \mathbf{F}_{2^m}[x]/g \)
so \( g \) divides \( F' \) in \( \mathbf{F}_{2^m}[x] \).
Proof: Assume
\[ \sum_i c_i/(x - a_i) = 0 \] in \( \mathbb{F}_{2^m}[x]/g \).

Write \( F = \prod_{i: c_i \neq 0} (x - a_i) \).

Then \( F'/F = \sum_{i: c_i \neq 0} 1/(x - a_i) \)
so \( F'/F = \sum c_i/(x - a_i) \)
so \( F'/F = 0 \) in \( \mathbb{F}_{2^m}[x]/g \)
so \( g \) divides \( F' \) in \( \mathbb{F}_{2^m}[x] \).

\( F' \) is a square:

if \( F = \sum_j F_j x^j \) then
\[ F' = \sum_j jF_j x^{j-1} \]
\[ = \sum_{j \in 1+2\mathbb{Z}} jF_j x^{j-1} \]
\[ = (\sum_{j \in 1+2\mathbb{Z}} \sqrt{jF_j} x^{(j-1)/2})^2. \]
The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
n = 1024, m = 10, t = 50.
This is too small:
$\approx 2^{60}$ pre-quantum security.
The McEliece cryptosystem

Standardize integers \( n \geq 0; t \geq 2; m \geq 1 \) with \( 2^m \geq n \).

1978 McEliece example:
\( n = 1024, m = 10, t = 50 \).
This is too small:
\( \approx 2^{60} \) pre-quantum security.

\( n = 2048, m = 11, t = 32 \):
\( \approx 2^{87} \) pre-quantum security.
The McEliece cryptosystem

Standardize integers \( n \geq 0; \ t \geq 2; \ m \geq 1 \) with \( 2^m \geq n \).

1978 McEliece example:
\( n = 1024, \ m = 10, \ t = 50 \).
This is too small:
\( \approx 2^{60} \) pre-quantum security.

\( n = 2048, \ m = 11, \ t = 32 \):
\( \approx 2^{87} \) pre-quantum security.

\( n = 3408, \ m = 12, \ t = 67 \):
\( \approx 2^{146} \) pre-quantum security.
The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
$n = 1024$, $m = 10$, $t = 50$.
This is too small:
$\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$:
$\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$:
$\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$:
$\approx 2^{263}$ pre-quantum security.
Alice’s secrets: monic irreducible \( g \in \mathbb{F}_{2^m}[x] \) with \( \deg g = t \); distinct \( a_1, \ldots, a_n \in \mathbb{F}_{2^m} \).
Alice’s secrets: monic irreducible \( g \in \mathbb{F}_{2^m}[x] \) with \( \deg g = t \); distinct \( a_1, \ldots, a_n \in \mathbb{F}_{2^m} \).

Note that \( g(a_1) \cdots g(a_n) \neq 0 \).
Define \( \Gamma \) as before.
Alice’s secrets: monic irreducible \( g \in \mathbb{F}_{2^m}[x] \) with \( \deg g = t \); distinct \( a_1, \ldots, a_n \in \mathbb{F}_{2^m} \).

Note that \( g(a_1) \cdots g(a_n) \neq 0 \).
Define \( \Gamma \) as before.

Alice’s public key:
\( mt \times n \) matrix \( K \) over \( \mathbb{F}_2 \) such that \( \Gamma = \text{Ker} \ K \).
Alice’s secrets: monic irreducible $g \in \mathbb{F}_{2^m}[x]$ with $\text{deg } g = t$; distinct $a_1, \ldots, a_n \in \mathbb{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$.

Define $\Gamma$ as before.

Alice’s public key:

$mt \times n$ matrix $K$ over $\mathbb{F}_2$

such that $\Gamma = \text{Ker } K$.

Bob chooses random $e \in \mathbb{F}_2^n$

with $|e| = t$; sends $Ke$. 

Alice’s secrets: monic irreducible
\( g \in \mathbb{F}_{2^m}[x] \) with \( \deg g = t \);
distinct \( a_1, \ldots, a_n \in \mathbb{F}_{2^m} \).

Note that \( g(a_1) \cdots g(a_n) \neq 0 \).
Define \( \Gamma \) as before.

Alice’s public key:
\( mt \times n \) matrix \( K \) over \( \mathbb{F}_2 \)
such that \( \Gamma = \ker K \).

Bob chooses random \( e \in \mathbb{F}_2^n \)
with \( |e| = t \); sends \( Ke \).

Alice receives \( Ke \),
finds \( v \in \mathbb{F}_2^n \) with \( Kv = Ke \),
decodes \( v \) to find \( v - e \).
1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbb{F}_2^n$ with $|e| = t$; sends $c + e$. 
1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbb{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify $\Gamma$ by an $(n - mt) \times n$ generator matrix $G$. 
1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbb{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify $\Gamma$ by an $(n - mt) \times n$ generator matrix $G$.

1986 Niederreiter improvements:

Send $Ke$ instead of $c + e$. 

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify $\Gamma$ by an $(n - mt) \times n$ generator matrix $G$.

1986 Niederreiter improvements:

Send $Ke$ instead of $c + e$.

$K$ is smaller than $G$ whenever $mt < n - mt$.

Compress $K$ to $mt(n - mt)$ bits by requiring systematic form.
Does structure of $\Gamma$ help attacker decrypt—
e.g., compute $g, a_1, \ldots, a_n$?
Does structure of $\Gamma$ help attacker decrypt—e.g., compute $g, a_1, \ldots, a_n$?

All known “structural attacks” are much slower than information-set decoding.
(Less conservative variants of McEliece encourage research.)
Does structure of $\Gamma$ help attacker decrypt—
e.g., compute $g, a_1, \ldots, a_n$?

All known “structural attacks” are much slower than information-set decoding.
(Less conservative variants of McEliece encourage research.)

Does $K$ leak more than $\Gamma$?
Does structure of $\Gamma$ help attacker decrypt—e.g., compute $g, a_1, \ldots, a_n$?

All known “structural attacks” are much slower than information-set decoding. (Less conservative variants of McEliece encourage research.)

Does $K$ leak more than $\Gamma$?

No with 1978 McEliece: matrix is explicitly randomized.
Does structure of $\Gamma$ help attacker decrypt—e.g., compute $g, a_1, \ldots, a_n$?

All known “structural attacks” are much slower than information-set decoding. (Less conservative variants of McEliece encourage research.)

Does $K$ leak more than $\Gamma$?

No with 1978 McEliece: matrix is explicitly randomized.

No with 1986 Niederreiter: matrix has systematic form.
Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.
Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Our constant-time software for batches of 256 decodings:

26544 Ivy Bridge cycles for \((n, t) = (2048, 32)\); \( \approx 2^{87} \).

79715 Ivy Bridge cycles for \((n, t) = (3408, 67)\); \( \approx 2^{146} \).

306102 Ivy Bridge cycles for \((n, t) = (6960, 119)\); \( \approx 2^{263} \).
The additive FFT

Fix \( n = 4096 = 2^{12}, \ t = 41. \)

Big final decoding step
is to find all roots in \( \mathbf{F}_{2^{12}} \)
of \( F = F_{41}x^{41} + \cdots + F_0x^0. \)

For each \( \alpha \in \mathbf{F}_{2^{12}}, \)
compute \( F(\alpha) \) by Horner’s rule:
41 adds, 41 mults.
The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step is to find all roots in $\mathbb{F}_{2^{12}}$ of $F = F_{41}x^{41} + \cdots + F_0x^0$.

For each $\alpha \in \mathbb{F}_{2^{12}}$, compute $F(\alpha)$ by Horner’s rule: 41 adds, 41 mults.

Or use “Chien search”: compute $F_i\gamma^i, F_i\gamma^{2i}, F_i\gamma^{3i}$, etc. Cost per point: again 41 adds, 41 mults.
The additive FFT

Fix \( n = 4096 = 2^{12}, t = 41 \).

Big final decoding step
is to find all roots in \( \mathbf{F}_{2^{12}} \)
of \( F = F_{41}x^{41} + \cdots + F_0x^0 \).

For each \( \alpha \in \mathbf{F}_{2^{12}} \),
compute \( F(\alpha) \) by Horner’s rule:
41 adds, 41 mults.

Or use “Chien search”: compute
\( F_i\gamma^i, F_i\gamma^{2i}, F_i\gamma^{3i}, \) etc. Cost per point: again 41 adds, 41 mults.

Our cost: 6.01 adds, 2.09 mults.
Asymptotics:
normally $t \in \Theta(n/\lg n)$,
so Horner’s rule costs
$\Theta(nt) = \Theta(n^2/\lg n)$. 
Asymptotics:

normally $t \in \Theta(n/ \log n)$, so Horner’s rule costs

$\Theta(nt) = \Theta(n^2 / \log n)$.

Wait a minute.

Didn’t we learn in school that FFT evaluates

an $n$-coeff polynomial

at $n$ points

using $n^{1+o(1)}$ operations?

Isn’t this better than $n^2 / \log n$?
Standard radix-2 FFT:

Want to evaluate
\[ F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1} \]
at all the \( n \)th roots of 1.

Write \( F \) as \( F_0(x^2) + xF_1(x^2) \).
Observe big overlap between
\[ F(\alpha) = F_0(\alpha^2) + \alpha F_1(\alpha^2) , \]
\[ F(-\alpha) = F_0(\alpha^2) - \alpha F_1(\alpha^2) . \]

\( F_0 \) has \( n/2 \) coeffs;
evaluate at \((n/2)\)nd roots of 1
by same idea recursively.
Similarly \( F_1 \).
Useless in char 2: $\alpha = -\alpha$.

Standard workarounds are painful. FFT considered impractical.


1996 von zur Gathen–Gerhard: some improvements.

2010 Gao–Mateer: much better additive FFT.

We use Gao–Mateer, plus some new improvements.
Gao and Mateer evaluate
\[ F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1} \]
on a size-\( n \) \( \mathbb{F}_2 \)-linear space.

Main idea: Write \( F \) as
\[ F_0(x^2 + x) + x F_1(x^2 + x). \]

Big overlap between \( F(\alpha) = F_0(\alpha^2 + \alpha) + \alpha F_1(\alpha^2 + \alpha) \)
and \( F(\alpha + 1) = F_0(\alpha^2 + \alpha) + (\alpha + 1) F_1(\alpha^2 + \alpha). \)

“Twist” to ensure \( 1 \in \text{space}. \)
Then \( \{ \alpha^2 + \alpha \} \) is a
size-\((n/2)\) \( \mathbb{F}_2 \)-linear space.
Apply same idea recursively.
We generalize to
\[ F = F_0 + F_1 x + \cdots + F_t x^t \]
for any \( t < n \).

⇒ several optimizations, 
not all of which are automated 
by simply tracking zeros.

For \( t = 0 \): copy \( F_0 \).

For \( t \in \{1, 2\} \):
\( F_1 \) is a constant.
Instead of multiplying 
this constant by each \( \alpha \), 
multiply only by generators 
and compute subset sums.
Syndrome computation

Initial decoding step: compute

\[ s_0 = r_1 + r_2 + \cdots + r_n, \]
\[ s_1 = r_1 \alpha_1 + r_2 \alpha_2 + \cdots + r_n \alpha_n, \]
\[ s_2 = r_1 \alpha_1^2 + r_2 \alpha_2^2 + \cdots + r_n \alpha_n^2, \]
\[ \vdots \]
\[ s_t = r_1 \alpha_1^t + r_2 \alpha_2^t + \cdots + r_n \alpha_n^t. \]

\( r_1, r_2, \ldots, r_n \) are received bits scaled by Goppa constants.
Typically precompute matrix mapping bits to syndrome.
Not as slow as Chien search but still \( n^{2+o(1)} \) and huge secret key.
Compare to multipoint evaluation:

\[ F(\alpha_1) = F_0 + F_1 \alpha_1 + \cdots + F_t \alpha_1^t, \]
\[ F(\alpha_2) = F_0 + F_1 \alpha_2 + \cdots + F_t \alpha_2^t, \]
\[ \vdots \]
\[ F(\alpha_n) = F_0 + F_1 \alpha_n + \cdots + F_t \alpha_n^t. \]
Compare to multipoint evaluation:

\[ F(\alpha_1) = F_0 + F_1 \alpha_1 + \cdots + F_t \alpha_1^t, \]
\[ F(\alpha_2) = F_0 + F_1 \alpha_2 + \cdots + F_t \alpha_2^t, \]
\[ \vdots \]
\[ F(\alpha_n) = F_0 + F_1 \alpha_n + \cdots + F_t \alpha_n^t. \]

Matrix for syndrome computation
is transpose of
matrix for multipoint evaluation.
Compare to multipoint evaluation:

\[ F(\alpha_1) = F_0 + F_1 \alpha_1 + \cdots + F_t \alpha_1^t, \]
\[ F(\alpha_2) = F_0 + F_1 \alpha_2 + \cdots + F_t \alpha_2^t, \]
\[ \vdots \]
\[ F(\alpha_n) = F_0 + F_1 \alpha_n + \cdots + F_t \alpha_n^t. \]

Matrix for syndrome computation is transpose of matrix for multipoint evaluation.

Amazing consequence: syndrome computation is as few ops as multipoint evaluation. Eliminate precomputed matrix.
Transposition principle:
If a linear algorithm computes a matrix $M$
then reversing edges and exchanging inputs/outputs computes the transpose of $M$.

1956 Bordewijk;
independently 1957 Lupanov for Boolean matrices.

1973 Fiduccia analysis:
preserves number of mults;
preserves number of adds plus number of nontrivial outputs.
We built transposing compiler producing C code.
Too many variables for $m = 13$; gcc ran out of memory.
We built transposing compiler producing C code. 
Too many variables for \( m = 13 \); gcc ran out of memory.

Used qhasm register allocator to optimize the variables.
Worked, but not very quickly.
We built transposing compiler producing C code. 
Too many variables for \( m = 13; \) 
gcc ran out of memory. 
Used qhasm register allocator to optimize the variables. 
Worked, but not very quickly. 
Wrote faster register allocator. 
Still excessive code size.
We built transposing compiler producing C code. 
Too many variables for \( m = 13; \) 
gcc ran out of memory.

Used qhasm register allocator to optimize the variables. 
Worked, but not very quickly.

Wrote faster register allocator. 
Still excessive code size.

Built new interpreter, 
allowing some code compression. 
Still big; still some overhead.
Better solution: stared at additive FFT, wrote down transposition with same loops etc.
Small code, no overhead.
Speedups of additive FFT translate easily to transposed algorithm.
Further savings: merged first stage with scaling by Goppa constants.
Results

60493 Ivy Bridge cycles:
  8622 for permutation.
20846 for syndrome.
  7714 for BM.
14794 for roots.
  8520 for permutation.

Code will be public domain.
We’re still speeding it up.

More information:
cr.yp.to/papers.html#mcbits