Code-based cryptography & brute-forcing McEliece keys

Lorenz Panny

Technische Universität München

11th Conference of the Fachgruppe Computeralgebra
Leipzig, 3 June 2025

Plan for this talk

- Code-based post-quantum cryptography.
- Code-based post-quantum cryptography.
- Code-based post-quantum cryptography.
- McEliece's public-key encryption scheme.
- Sendrier's support-splitting algorithm (SSA).
- Non-uniqueness of private keys in McEliece.
- Fast implementation techniques for key search.
- Results & summary.

Public-key cryptography

...refers to cryptography in which different levels of knowledge enable users to perform different operations. (See examples next slides.)

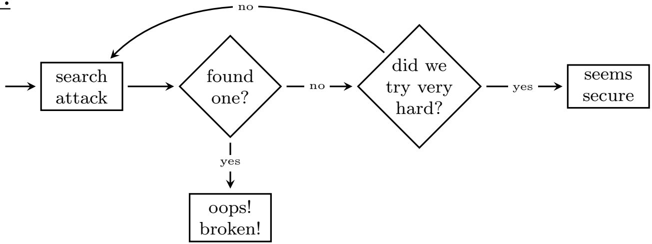
🎉 Almost always based on well-behaved algebraic structures.

Groups, rings, group actions, lattices, codes, ...

⚠️ It is unknown whether public-key cryptography *exists*.

(If it does, then $\mathbf{P} \neq \mathbf{NP}$.)

<u>Reality:</u>



Example: Public-key encryption

- **Anyone** can use Bob's public key to encrypt a message.
- **Bob** can decrypt it using his private key.
- **Noone but Bob** can learn anything about the *message*.

  (except the length)

Analogy: An open padlock for which *Bob has the key*.

Example: Digital signatures

- **Alice** uses her private key to sign a *message*.
- **Anyone** can verify the *signature* using Alice's public key.
- **Noone but Alice** can forge a valid *signature* for a new *message*.

 This mimics the *intended* properties of a "real" (analog) signature.

Plan for this talk

- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography.
- Code-based post-quantum cryptography.
- McEliece's public-key encryption scheme.
- Sendrier's support-splitting algorithm (SSA).
- Non-uniqueness of private keys in McEliece.
- Fast implementation techniques for key search.
- Results & summary.

The quantum threat

...is a major issue for public-key cryptography in particular.

Today's most popular public-key schemes are based on:

- The presumed hardness of factoring large integers.
- The presumed hardness of computing discrete logarithms.

(The discrete-logarithm problem in a group $\langle g \rangle$ is to invert the map $x \mapsto g^x$.)

<u>Shor (1994):</u> Polynomial-time quantum algorithms for both!

However, not all hope is lost:

$\exists$ plenty of apparently quantum-hard problems.

$\rightsquigarrow$ *<u>P</u>ost-quantum <u>c</u>ryptography (PQC)*

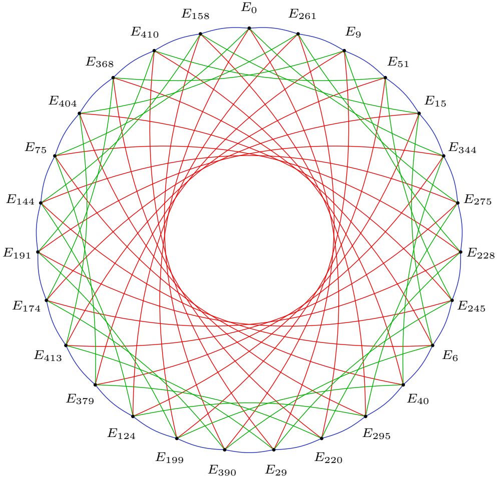
Based on different sources of hard problems:

Isogenies between abelian varieties❤, (structured) lattices, codes, multivariate systems, symmetric cryptography, ...

Digression: Isogeny-based cryptography

...is what I've been doing most of the time.

Ask me about it later. :)



Plan for this talk

- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography.
- McEliece's public-key encryption scheme.
- Sendrier's support-splitting algorithm (SSA).
- Non-uniqueness of private keys in McEliece.
- Fast implementation techniques for key search.
- Results & summary.

(Linear) codes

Wall of definitions:

- An $[n, k]$ code C over $\mathbb{F}_q$ is a k-dimensional subspace of $\mathbb{F}_q^n$.
  A generator matrix of C is any $G \in \mathbb{F}_q^{k\times n}$ such that $\mathbb{F}_q^k G = C$.
- We equip $\mathbb{F}_q^n$ with the Hamming weight: The number of nonzero coefficients. It induces the Hamming distance.
- Codes can equivalently be described using a parity-check matrix: That is, a $H \in \mathbb{F}_q^{(n-k)\times n}$ satisfying $GH^T = 0$.
- Isomorphisms of codes are (Hamming) isometries.
  They are $C \mapsto CP$ with $P \in \mathrm{GL}_n(\mathbb{F}_q)$ a monomial matrix.

(Monomial matrix = permutation matrix · full-rank diagonal matrix.)

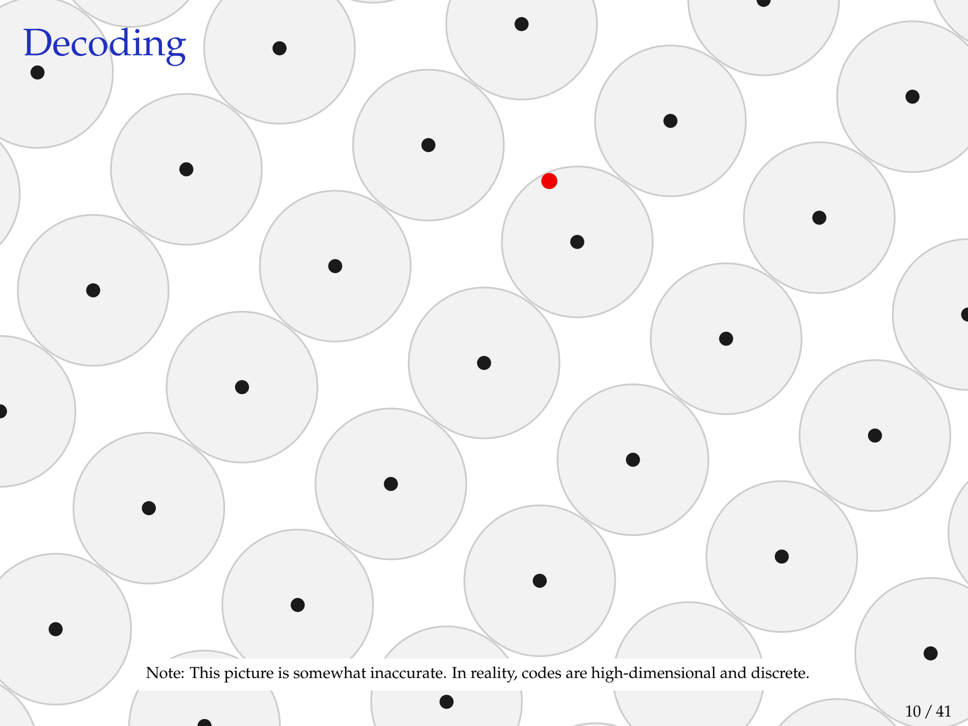
(For $q = 2$, these are just permutation matrices.)

Cryptography from linear codes

Traditional purpose of linear codes: Error correction.

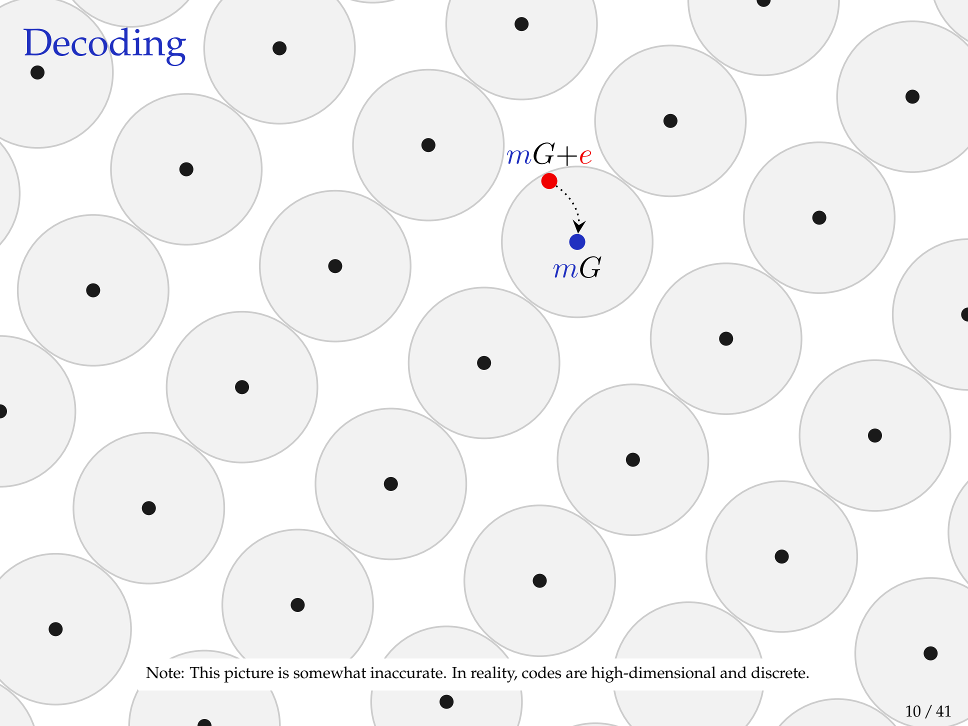
- Encoding: Represent a *message* $m \in \mathbb{F}_q^k$ as the *code word* $mG \in \mathbb{F}_q^n$.
- Decoding: Compute m from $mG + e$ where $e \in \mathbb{F}_q^n$ is low-weight *error*.

Decoding



Note: This picture is somewhat inaccurate. In reality, codes are high-dimensional and discrete.

Decoding



Note: This picture is somewhat inaccurate. In reality, codes are high-dimensional and discrete.

Cryptography from linear codes

Traditional purpose of linear codes: Error correction.

- Encoding: Represent a *message* $m \in \mathbb{F}_q^k$ as the *codeword* $mG \in \mathbb{F}_q^n$.
- Decoding: Compute m from $mG + e$ where $e \in \mathbb{F}_q^n$ is low-weight *error*.

Decoding is generally hard for random codes.

$\rightsquigarrow$ Blueprint for public-key cryptography:

- Alice generates an easily decodable code from some suitable family.
- Alice "scrambles" the code into a random-*looking* code and publishes it.

$\Longrightarrow$ Anyone can encode, only Alice can decode.

"Scrambling": Apply a random isometry & sample a random generator matrix. That is, let $\widehat{G} := SGP$ with $S \in \mathrm{GL}_n$ and P an isometry.

Assumption: The map $G \mapsto \widehat{G}$ is one-way. $\rightsquigarrow$ *Cryptography!*

Attack strategies

There are two main assumptions an attacker could try to break:

- Try to decode directly on the public, random-looking code.
  This is the "decoding attack". $\rightsquigarrow$ Next slide.
- Try to recover the hidden secret code from the public code.
  This is the "key-recovery attack". $\rightsquigarrow$ Rest of the talk.

Information-set decoding (ISD)

...is the dominant family of generic decoding algorithms.

Main idea: Guess that certain parts of the codeword are error-free, solve using linear algebra.

For $H \in \mathbb{F}_q^{(n-k)\times n}$ a parity-check matrix and $c = mG + e \in \mathbb{F}_q^n$:

- Pick a random permutation matrix $P \in \mathbb{F}_q^{n\times n}$.
- Bring HP to echelon form $H' = UHP$. (Assume $H' = (\mathbf{1} \mid Q)$ with $Q \in \mathbb{F}_q^{(n-k)\times k}$.)
- Pray that $P^{-1}e$ is of the form $(s' \parallel \mathbf{0})$ with $s' \in \mathbb{F}_q^{n-k}$.
- If it is, then $H'P^{-1}c = UHc = UHe = H'P^{-1}e = H'(s' \parallel \mathbf{0}) = s'$.
  (This case can usually be detected by checking $\mathrm{wt}(s')$: It should be small.)

$\Longrightarrow$ We can find e as $P(H'P^{-1}c \parallel \mathbf{0}) = P(s' \parallel \mathbf{0})$, then solve $mG = c - e$ for m.

$$\rightsquigarrow \Pr[\mathit{success}] = \binom{n-k}{t} / \binom{n}{t} \text{ where } t = \mathrm{wt}(e).$$

- The above is a very basic variant of ISD [Prange 1962].
- $\exists$ plenty of improvements with better complexity.
- For well-chosen codes and $\mathrm{wt}(e)$, still exponential-time.

Key-recovery attacks

...are much more expensive for well-chosen families of codes.
Example: For "Classic McEliece", decoding is $2^{hundreds}$ while key recovery is $2^{thousands}$.

Contrary to decoding, the details depend on the specific family of codes under consideration.

Historically, key recovery has (arguably) been much less well-understood than decoding.

Nowadays, this is changing.

- New algebraic distinguishers for Goppa codes.
- New concrete cost estimates for McEliece key recovery.
  That is: *How expensive is "smart brute force", really?*

Plan for this talk

- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- McEliece's public-key encryption scheme.
- Sendrier's support-splitting algorithm (SSA).
- Non-uniqueness of private keys in McEliece.
- Fast implementation techniques for key search.
- Results & summary.

McEliece's encryption scheme

...is a straightforward instantiation of the code-based blueprint to make a public-key encryption scheme.

(Recall: This means anyone can encrypt, but only the intended recipient can decrypt.)

- Proposed in 1978 (!) by Robert J. McEliece.
- Original suggestion: Use (binary) Goppa codes.
- Current state of the art: Use (binary) Goppa codes.
- Initially unpopular for its large key sizes ($\geq$ hundreds of kB).
- Nowadays, much more popular for its (conjectured) post-quantum security and stable security history.

Goppa codes

- Parameters: Prime power $q = p^m$ and $t, n \in \mathbb{Z}_{\geq 1}$ with $tm \leq n \leq q$.
- Data: – Monic irreducible polynomial $g \in \mathbb{F}_q[x]$ of degree t.
  – Sequence $L = (\alpha_1, \ldots, \alpha_n)$ of distinct elements of $\mathbb{F}_q$.
  (Assume $g(\alpha_i) \neq 0$ for all i.)

$\rightsquigarrow$ Code $\Gamma(g, L) := \{\, c \in \mathbb{F}_p^n \;:\; \sum_{i=1}^n c/(x - \alpha_i) \equiv 0 \pmod{g} \,\}$.

(Dimension $\geq n - tm$, distance $\geq 2t + 1$. (Assume equality throughout.))

$\rightsquigarrow$ Parity-check matrix (identifying $\mathbb{F}_q = \mathbb{F}_p^m$ as $\mathbb{F}_p$-vector spaces):

$$H = \begin{bmatrix} {\alpha_1}^0/g(\alpha_1) & \cdots & {\alpha_n}^0/g(\alpha_n) \\ \hline {\alpha_1}^1/g(\alpha_1) & \cdots & {\alpha_n}^1/g(\alpha_n) \\ \hline \vdots & \ddots & \vdots \\ \hline {\alpha_1}^{t-1}/g(\alpha_1) & \cdots & {\alpha_n}^{t-1}/g(\alpha_n) \end{bmatrix} \in \mathbb{F}_p^{tm \times n}$$

$\Longrightarrow$ To sample a Goppa code, pick g and L.

“Scrambling” Goppa codes

In practice:

The “scrambled” version of a Goppa code is simply given by the echelon form of H. (Good for simplicity & efficiency!)

Earlier:

“Scrambling” is $G \mapsto \widehat{G} = SGP$ with $S \in \mathrm{GL}_n$ and P a monomial matrix.

Q: Where did S and P go?

A: For $\widehat{G} = SGP$ we get $\widehat{H} = HP^{-T}$.

- For $p = 2$ the choice of P disappears in the choice of the α_i.
  (Over $\mathbb{F}_2$, monomial matrices are just permutation matrices.)
- The echelon form is a worst-case basis of the row span.
  Reasoning: For any other choice of basis, the attacker can always just compute the echelon form on their own and thus reduce to this case in polynomial time.

Plan for this talk

- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- McEliece's public-key encryption scheme. ✓
- Sendrier's support-splitting algorithm (SSA).
- Non-uniqueness of private keys in McEliece.
- Fast implementation techniques for key search.
- Results & summary.

Reducing the search space

Observation: For n not much smaller than q, most of the size of the private-key space $\{(g, L)\}$ comes from the permutation of L.

Can we guess $\mathsf{set}(L)$ instead of L?

(For $\vec{v} = (v_1, \dots, v_n) \in S^n$, we write $\mathsf{set}(\vec{v}) := \{v_1, \dots, v_n\}$.)

What's needed is a solver for permutation equivalence.

Two codes $C, C' \subseteq \mathbb{F}_q^{k \times n}$ are called permutation-equivalent if there exists a permutation matrix $P \in \mathbb{F}_q^{n \times n}$ such that $C' = CP$. Note that P is an isometry.

Generally: ?

Practically: Usually efficient!

:-) Sendrier (2000): The support-splitting algorithm (SSA) can decide if P exists and, if so, find it.

(Not much has been *proven* about this algorithm. In practice, it is very fast.)

Splitting the support

Assumption: Have permutation invariant $\mathcal{V}$ on codes that is...:

... efficiently computable.

... discriminant, i.e., likely to take distinct values on inequivalent codes.

Now suppose C, C' are permutation-equivalent, i.e., $C' = CP$.

Guess that P maps i to j, then puncture C at i and C' at j and check if they can *still* be equivalent by evaluating $\mathcal{V}$.

(Puncturing at i means projecting to $\mathbb{F}_q^{i-1} \times \{0\} \times \mathbb{F}_q^{n-i-1}$.)

$\rightsquigarrow$ Yes: P might map i to j. Continue guessing more positions.

$\rightsquigarrow$ No: P cannot map i to j. Backtrack and continue with a different guess.

The support-splitting algorithm is a streamlined variant of this.

- Instead of guessing blindly, puncture out entire sets J of positions for which $\mathcal{V}$ has previously behaved identically.
- Then, hopefully, the hulls of singly-punctured codes $C_{J\cup\{j\}}$ for varying j will refine the partition some more. (Same for C'.)

Hull enumerators

Q: How to construct a suitable permutation invariant $\mathcal{V}$?

A (version 0): Use the enumerator of a code. This is the vector $\mathcal{W}(C) := (w_0, w_1, ..., w_n) \in \mathbb{Z}^n_{\geq 0}$ where $w_i = \{c \in C : \text{wt}(c) = i\}$.

☹ Best algorithm seems to be to enumerate all codewords.

(Honorable mention: Gray code.)

A (version 1): Use the enumerator of the *hull* of a code.
The hull is $C \cap C^\perp$ where $C^\perp = \{c' \in \mathbb{F}_q^n : \forall c \in C. \langle c, c' \rangle = 0\}$.

☺ It is compatible with permutations and low-dimensional!

(Proportion of n-dimensional codes over $\mathbb{F}_q$ with hull dimension ℓ is $\approx C/q^{\ell(\ell+1)/2}$ where $0.419 < C < 1$.)

Empirically, the hull enumerator makes SSA work very well!

Splitting the support, quickly

Main algorithmic ingredients for computing hull enumerators:

- Largest effort: Gauß-esque echelon-form computation.
- *Cool trick* (Sendrier 2000) for computing all singly-punctured hulls from a single row-reduced basis matrix.
- Enumeration of hull vectors, tallying Hamming weights.

\+ Lots of general algorithmic bookkeeping: Tracking partitions of $\{1, \ldots, n\}$, codes punctured at various locations, etc.

☹ All of this is a bit annoying to implement *fast*:

- ☹ Variable-sized data structures!
- ☹ Dynamic memory allocations!
- ☹ Unpredictable execution flow!
- ☹ Unpredictable memory-access patterns!

However, stay tuned. ☺

Plan for this talk

- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- McEliece's public-key encryption scheme. ✓
- Sendrier's support-splitting algorithm (SSA). ✓
- Non-uniqueness of private keys in McEliece.
- Fast implementation techniques for key search.
- Results & summary.

How many Goppa codes are there? (1)

Naïve count:

- There are $\approx q^t/t$ monic irreducible $g \in \mathbb{F}_q[x]$ of degree t.
- There are $q!/(q-n)!$ choices for L.

$\rightsquigarrow$ Total count $\approx \frac{q^t \cdot q!}{t \cdot (q-n)!}$.

Count modulo permutation equivalence:

- There are $\approx q^t/t$ monic irreducible $g \in \mathbb{F}_q[x]$ of degree t.
- There are $\binom{q}{n}$ choices for $\mathsf{set}(L) = \{\alpha_1, ..., \alpha_n\}$.

$\rightsquigarrow$ Total count $\approx \frac{q^t \cdot \binom{q}{n}}{t}$.

!! This formula still *overestimates* the number of Goppa codes.

How many Goppa codes are there? (2)

Definition: The affine semilinear group of $\mathbb{F}_q$ is the subgroup

$$\mathrm{A\Gamma L}(q) := \{\, (x \mapsto Ax^{\varphi} + B) \;:\; A \in \mathbb{F}_q^{\times},\, B \in \mathbb{F}_q,\, \varphi \in \mathrm{Aut}(\mathbb{F}_q) \,\}$$

of $\mathrm{Sym}(\mathbb{F}_q)$. (Equivalently: $\mathbb{F}_q^{\times} \times \mathbb{F}_q \times \mathrm{Aut}(\mathbb{F}_q)$ with a funny composition law.)

Consider group actions $*$ of $\mathrm{A\Gamma L}(q)$:

- On $\mathbb{F}_q^n$ by coordinate-wise application.
- On monic polynomials over $\mathbb{F}_q$ by applying $x \mapsto Ax^{\Phi} + B$ to all *roots* of the polynomial, where $\Phi \in \mathrm{Aut}(\overline{\mathbb{F}_q})$ is a lift of φ.
  (Well-definedness: (1) The result is defined over $\mathbb{F}_q$; (2) Choices of Φ differ only by $\mathrm{Gal}(\overline{\mathbb{F}_q}/\mathbb{F}_q)$, hence merely permute the roots, leaving the polynomial invariant.)

Theorem: For any $\tau \in \mathrm{A\Gamma L}(q)$ and any pair g, L defining a Goppa code, we have

$$\Gamma(\tau * g, \tau * L) = \Gamma(g, L)\,.$$

[Probably folklore/known to experts. Previous literature: Moreno 1979 ($p{=}2$, $|L|{=}q$), Gibson 1991 (cryptanalytic application), etc.]

How many Goppa codes are there? (3)

$\rightsquigarrow$ The private key (g, L) is non-unique in McEliece!

$\Longrightarrow$ Searching for the pair (g, L) using brute force succeeds faster than a naïve estimate suggests.

Previous estimate: About $q^t \binom{q}{n} / t$ guesses.

Equivalences from $\mathrm{A\Gamma L}(q)$: About $|\mathrm{A\Gamma L}(q)| = q(q-1)m$ private keys per public key.

Updated **estimate**: About $q^t \binom{q}{n} / \big(tq(q-1)m\big)$ guesses.

☹ This formula can still *over- and undercount* Goppa codes.

- Some $\mathrm{A\Gamma L}$ equivalences might *already* be explained by permutation equivalence: When $\tau * g = g$ and $\mathsf{set}(\tau * L) = \mathsf{set}(L)$ for all $\tau \in \mathrm{A\Gamma L}(q)$.
- There may be permutation equivalences that *aren't* explained by $\mathrm{A\Gamma L}$.

☺ Luckily, both effects are rare for "non-small" parameters.

$\Longrightarrow$ The **estimate** above is practically good enough.

Plan for this talk

- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- McEliece's public-key encryption scheme. ✓
- Sendrier's support-splitting algorithm (SSA). ✓
- Non-uniqueness of private keys in McEliece. ✓
- Fast implementation techniques for key search.
- Results & summary.

Splitting the support, many times

Observation #1: In the context of McEliece key search, one of the codes given to the SSA remains fixed throughout.

⇝ can precompute lots of data about the target public key.

Observation #2: In the context of McEliece key search, it suffices to recognize inequivalent codes quickly.
(Few *possibly* equivalent codes can be checked again using a second, perfectly correct test.)

⇝ can trade correctness for speed.

Fast filtering

Find "characteristic" singly-punctured hull enumerators:

- Enumerators that appear for the target public-key code, but are unlikely to appear for a random code.
- Enumerators that do not appear for the target public-key code, but are likely to appear for a random code.

$\rightsquigarrow$ A "fast filter" $\mathcal{F}$ for a target public-key code C is a list of such enumerators such that $\Pr[pass] = \varepsilon$ for a random code.

:-) Sometimes, almost all <u>in</u>equivalent codes can be quickly discarded by checking for the presence of a *single* punctured hull enumerator.

Everything is a binary circuit

Turn the entire "fast filter" into a binary circuit.

☺ No more complicated data structures, predictable execution flow & memory-access patterns, flexible choice between (simpler & faster & more energy-efficient) hardware platforms, ...

☹ Things like memory access and integer arithmetic can be emulated using bit operations, but this is much more expensive than using the CPU's silicon implementations of the same operations.

Computing hulls, quickly

Variant of reduced echelon form: “diagonal standard form”.

[Sendrier 2000]

$$\begin{bmatrix}
\mathbf{1} & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & \mathbf{1} & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \mathbf{1} & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}
\end{bmatrix}$$

:) This can be computed using an algorithm that is:

- branch-free: Fixed sequence of logical operations.

  ⇝ easily circuit-able!

- restartable: Can reuse previous work after column update.

  ⇝ To search for $L = (\alpha_1, \ldots, \alpha_n)$, we can replace elements α_i one at a time!

  ⇝ Track partially reduced matrices for prefixes of L in a stack data structure.

Lemma 17. *Let C be a code given by a square matrix $M \in \mathbb{F}_q^{n\times n}$ in diagonal standard form. Then the hull $C \cap C^\perp$ equals the (right) kernel of $\mathbb{1} + M - M^\top$.*

[“Diagonal standard form” goes back to Sendrier (2000). The circuit abstraction & reusing linear-algebra work are new in this context.]

Gauß-esque elimination as a circuit

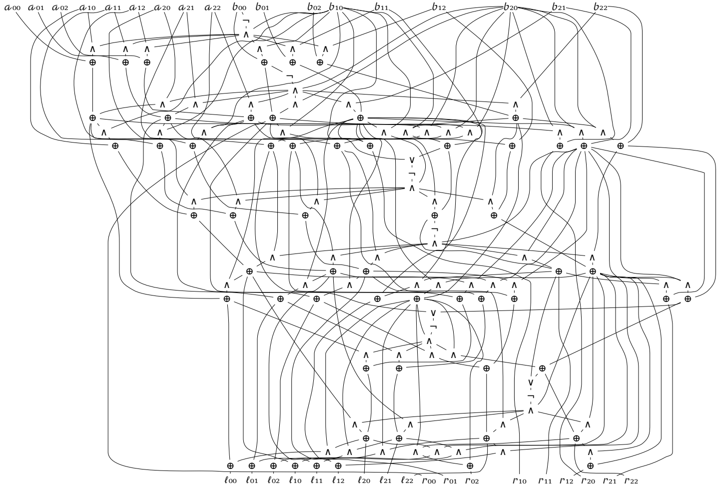
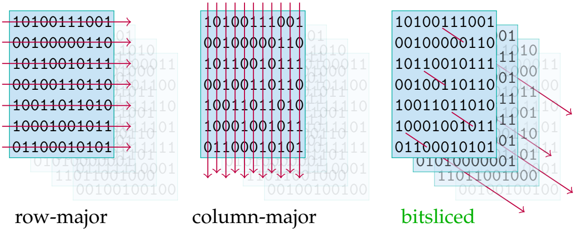


Illustration: Binary circuit to compute the diagonal standard form and a transformation matrix for a given matrix, in the (very small) case $n = 3$.

Splitting many supports, simultaneously

Goal: Execute a “fast filtering” circuit many times (in parallel) on a large set of different inputs.

Packing a collection of matrices over $\mathbb{F}_2$ into CPU registers:



Good idea: Use the bitsliced representation.

- Every w-bit register holds a single bit from w separate instances.

⇝ Predictable execution flow and memory-access pattern.

(It *should* also be a good idea to use wider vector registers rather than general-purpose CPU registers, but some quick and dirty experiments indicated this to be slower on Zen 4c.)

Plan for this talk

- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- McEliece's public-key encryption scheme. ✓
- Sendrier's support-splitting algorithm (SSA). ✓
- Non-uniqueness of private keys in McEliece. ✓
- Fast implementation techniques for key search. ✓
- Results & summary.

TII's McEliece challenge instances

- 2023–2024: McEliece Challenges run by the TII Institute.
- Multiple categories (decoding/key recovery; theory/practice).
- Wide range of estimated security levels.
- Cash prizes for best solutions in each category.

I won :-) (in the "practical key recovery" category).

Technique: This talk, lovingly cast into 1770 lines of C++.

Current record: "83 bits"

- Challenge instance: $p = 2$, $m = 8$, $t = 5$, $n = 253$.

Parity-check matrix of public-key code:

- Naïve attack cost estimate: $(2^8)^5/5 \cdot \binom{2^8}{253} \cdot 253^3 \approx 2^{83.025}$.

(Here n^3 appears to be a rough estimate for the cost of the support-splitting algorithm.)

- Actual time spent: Only 1,735 CPU days. (Total $\approx 2^{58.}$– clock cycles.)

(Tested $\approx 2^{39}$ key guesses at a rate of $\approx$ 7,500 per core and second.)

(Newer version of software: Estimated 1,400 CPU days, testing $\approx$ 9,400 guesses per core and second.)

Estimates

instance	m	t	n	$\approx$ # guesses	# $\mathcal{F}$	$\approx \Pr[\mathcal{F} \mapsto \texttt{true}]$	guesses/(core · s)	$\approx$ core time
69	6	4	57	$2^{36.65}$	9	$2^{-15.23}$	$2^{18.71}$	$2^{17.94}$ s $\approx$ 2.9 d
70	8	5	255	$2^{26.68}$	23	$2^{-9.25}$	$2^{11.79}$	$2^{14.89}$ s $\approx$ 8.4 h
71	6	6	60	$2^{38.13}$	8	$2^{-16.20}$	$2^{18.64}$	$2^{19.49}$ s $\approx$ 8.5 d
72	7	5	125	$2^{34.26}$	1	$2^{-17.19}$	$2^{16.23}$	$2^{18.04}$ s $\approx$ 3.1 d
73	7	6	126	$2^{35.61}$	1	$2^{-23.79}$	$2^{16.07}$	$2^{19.54}$ s $\approx$ 8.8 d
74	7	8	128	$2^{36.20}$	20	$2^{-6.94}$	$2^{10.74}$	$2^{25.47}$ s $\approx$ 1.47 yr
76	6	7	60	$2^{43.91}$	3	$2^{-18.99}$	$2^{18.97}$	$2^{24.93}$ s $\approx$ 1.02 yr
77	7	5	124	$2^{39.23}$	4	$2^{-16.64}$	$2^{15.80}$	$2^{23.43}$ s $\approx$ 4.3 mo
78	6	8	61	$2^{45.78}$	3	$2^{-14.98}$	$2^{18.43}$	$2^{27.35}$ s $\approx$ 5.42 yr
79	7	6	125	$2^{41.00}$	4	$2^{-16.74}$	$2^{15.67}$	$2^{25.33}$ s $\approx$ 1.34 yr
80	7	7	126	$2^{42.39}$	2	$2^{-21.01}$	$2^{16.02}$	$2^{26.37}$ s $\approx$ 2.74 yr
81	7	8	127	$2^{43.20}$	4	$2^{-16.08}$	$2^{15.35}$	$2^{27.86}$ s $\approx$ 7.71 yr
82	6	8	60	$2^{49.72}$	3	$2^{-16.01}$	$2^{18.55}$	$2^{31.16}$ s $\approx$ 76.18 yr
83	8	5	253	$2^{40.08}$	1	$2^{-19.95}$	$2^{13.21}$	$2^{26.87}$ s $\approx$ 3.90 yr
84	8	6	254	$2^{41.42}$	20	$2^{-10.62}$	$2^{12.60}$	$2^{28.82}$ s $\approx$ 14.99 yr
85	8	8	256	$2^{42.01}$	20	$2^{-10.26}$	$2^{9.90}$	$2^{32.10}$ s $\approx$ 146.1 yr
86	7	5	122	$2^{48.22}$	1	$2^{-16.72}$	$2^{16.42}$	$2^{31.79}$ s $\approx$ 118.0 yr
87	7	8	126	$2^{49.19}$	4	$2^{-15.69}$	$2^{15.74}$	$2^{33.45}$ s $\approx$ 371.9 yr
88	7	9	127	$2^{50.03}$	23	$2^{-6.78}$	$2^{12.17}$	$2^{37.86}$ s $\approx$ 7,900 yr
89	8	5	252	$2^{46.06}$	1	$2^{-17.96}$	$2^{13.23}$	$2^{32.83}$ s $\approx$ 242.5 yr

Future work

- Conditions for the $q^t \binom{q}{n} / (tq(q-1)m)$ count to be accurate?
- More bit operations per unit of time: GPU, FPGA, ASIC?
- Exploit matrix symmetry in punctured-hull computation?
- Different approach to support splitting altogether?

Summary

- The McEliece key-recovery problem is a little bit easier than one might think.

- The impact on real parameters is effectively nonexistent.

  This is because decoding attacks have always been much cheaper, hence they are what primarily constrains the parameter choices.

  Example: "Classic McEliece" parameter set `348864` estimates $\geq 2^{140.8}$ operations for decoding, but a brute-force key-recovery attack requires $\geq 2^{3210.4}$ operations.

Plan for this talk

- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- Code-based post-quantum cryptography. ✓
- McEliece's public-key encryption scheme. ✓
- Sendrier's support-splitting algorithm (SSA). ✓
- Non-uniqueness of private keys in McEliece. ✓
- Fast implementation techniques for key search. ✓
- Results & summary. ✓

Questions?

Check out my preprint: `https://ia.cr/2025/632`

(Also feel free to email me: `lorenz@yx7.cc`)