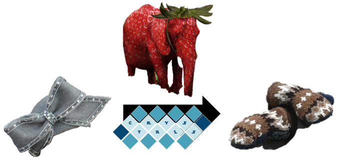


RSA is dead, long live PQC!
Teaching cryptography in the quantum era

Joshua Holden (he/him/his)

`http://www.rose-hulman.edu/~holden`

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Land Acknowledgement

This talk is being broadcast from land that is part of the traditional territories of the Očeti Šakówiŋ (Sioux), Kiikaapoi (Kickapoo), and Myaamia (Miami) nations. These peoples, and many others, are still fighting for the rights promised them by treaties with the United States government.

BIPOC lives and heritages matter.

What is Post-Quantum Cryptography?

Not . . .

- . . . "The thing that comes after Quantum Cryptography"
- . . . Quantum Key Distribution
- . . . Quantum Computation

Post-Quantum Cryptography (PQC) is cryptography that we can run on today's computers but which will be resistant to cryptanalysis by quantum computers.

In 2016, NIST estimated that a cryptographically relevant quantum computer could be built in 15 years.

- RSA: dead
- Diffie-Hellman Key Agreement: dead
- Digital Signature Algorithm: dead
- Elliptic Curve Cryptography: dead
- AES: still alive
- SHA-3: still alive (maybe even SHA-2)

Several types of cryptography will just need longer keys, but public-key cryptography will need a complete revamp.

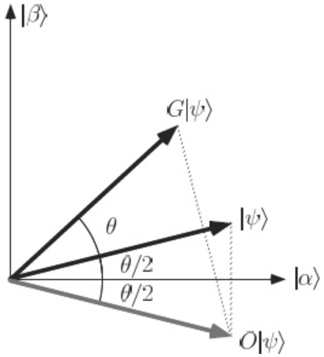
NIST started a process to choose new public-key systems for standardization.

- Submissions received by NIST: 82
- Submissions meeting minimum specified requirements: 69
- Submissions still in contention as of the First PQC Standardization Conference: 64
- Submitters involved: 278, from “25 Countries, 16 States, 6 Continents”

These systems are referred to as “post-quantum cryptography”, although quantum-resistant cryptography might be more accurate.

There are two main quantum algorithms relevant to cryptanalysis.

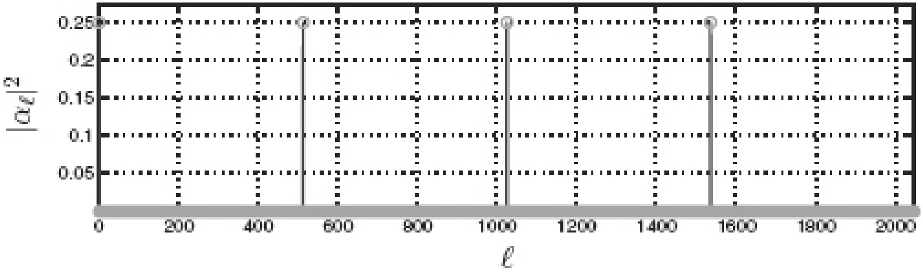
Grover's algorithm speeds up searching (e.g. for keys), but only by a quadratic amount.



Solution: Double the key size.

There are two main quantum algorithms relevant to cryptanalysis.

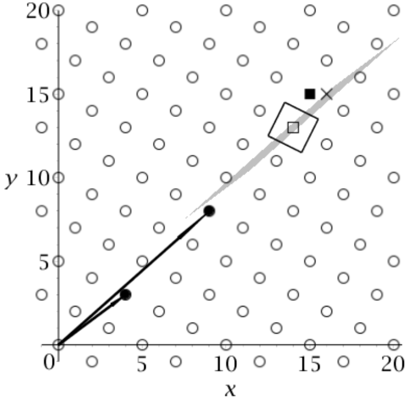
Shor's algorithm speeds up finding periodic patterns, by an exponential amount.

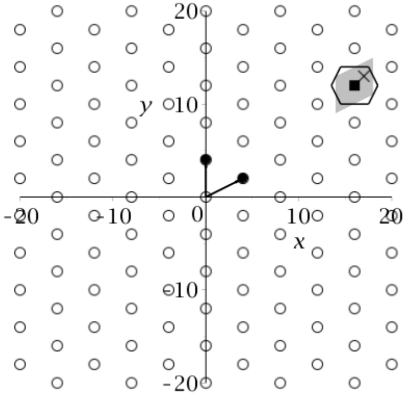


This completely breaks anything based on factoring or any variant of the discrete logarithm problem.

There are five major types of hard problems under consideration for post-quantum cryptography.

1. Lattice problems



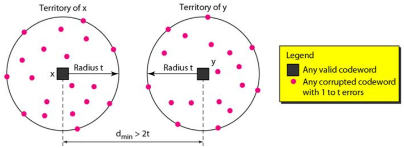


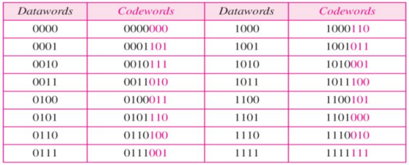
Aretha can encrypt a point using these generators and a small error.

Bernie can decrypt because he knows a better set of generators.

There are five major types of hard problems under consideration for post-quantum cryptography.

2. Code problems





Datawords	Codewords	Datawords	Codewords
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Aretha can encrypt a bitstring using these generators and a small error.

Bernie can decrypt because he knows a better set of generators.

There are five major types of hard problems under consideration for post-quantum cryptography.

3. Multivariable polynomial problems

$$\begin{aligned} y_1 &= f_1(x_1, \ldots, x_n) \\ y_2 &= f_2(x_1, \ldots, x_n) \\ &\vdots \\ y_m &= f_m(x_1, \ldots, x_n) \end{aligned}$$

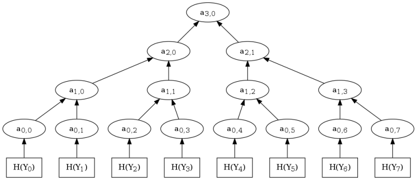
$$\begin{aligned} y_1 &= f_1(x_1, \ldots, x_n) \\ y_2 &= f_2(x_1, \ldots, x_n) \\ &\vdots \\ y_m &= f_m(x_1, \ldots, x_n) \end{aligned}$$

Aretha can solve these equations using a trap door $f = S \circ f^* \circ T$.

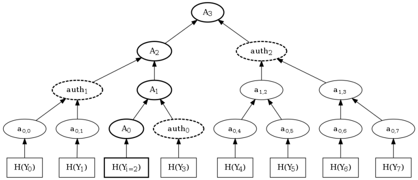
Bernie can verify that the solution is correct.

There are five major types of hard problems under consideration for post-quantum cryptography.

4. Hash function problems



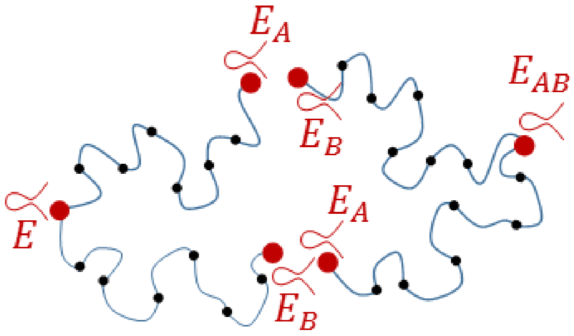
Aretha can sign a message because she knows all of the Y_i.



Bernie can verify the signature using one Y_i and the “auth” values.

There are five major types of hard problems under consideration for post-quantum cryptography.

5. Elliptic Curve Isogenies (some of these have recently been broken!)



Aretha can reach the secret curve by going around the top route.

Bernie can reach the secret curve by going around the bottom route.

NIST has so far selected four submissions for standardization.

- CRYSTALS-Kyber (key-establishment for most use cases)
- CRYSTALS-Dilithium (digital signatures for most use cases)
- FALCON (digital signatures for use cases requiring smaller signatures)
- SPHINCS+ (digital signatures not relying on the security of lattices)

The first three are lattice systems; the fourth is a hash function system.

CRYSTALS

Cryptographic Suite for Algebraic Lattices

Joppe Bos   Leo Ducas

Eike Kiltz   Tancrede Lepoint

Vadim Lyubashevsky   John Schanck

Peter Schwabe   Gregor Seiler   Damien Stehle

CRYSTALS-Kyber is based on a version of the Learning With Errors (LWE) problem.

Given a vector **t** of the form

$$\mathbf{t} \equiv A\mathbf{s} + \mathbf{e} \pmod{q}, \tag{1}$$

where

- *A* is a public matrix with at least as many rows as columns
- **e** is a “small error vector” drawn from some probability distribution

find the secret vector **s**.

Basic LWE key generation:

Bernie's private key is a matrix S of ℓ columns and his public key is (A, T), where

$$T \equiv AS + E \pmod{q} \tag{2}$$

for an error matrix E of the appropriate size.

Basic LWE encryption:

$\mathbf{p}$ is a plaintext message represented as a vector of bits.

Aretha chooses

- a random ℓ-bit nonce vector $\mathbf{r}$;
- small random errors $\mathbf{e}_1$ and $\mathbf{e}_2$.

She computes the ciphertext $(\mathbf{u}, \mathbf{v})$, where

$$\mathbf{u} = \left(A^T\mathbf{r} + \mathbf{e}_1\right) \text{MOD } q, \qquad \mathbf{v} = \left(T^T\mathbf{r} + \mathbf{e}_2 + \lfloor q/2 \rceil\, \mathbf{p}\right) \text{MOD } q.$$

Basic LWE decryption:

Bernie tests each coordinate of $\mathbf{v} - S^T\mathbf{u}$ to see if it is closer to 0 or to $q/2$ modulo q:

$$\mathbf{p}' = \left\lfloor \lfloor q/2 \rceil^{-1} (\mathbf{v} - S^T\mathbf{u}) \right\rceil \operatorname{MOD} 2.$$

Since

$$\begin{aligned} \mathbf{v} - S^T\mathbf{u} &\equiv T^T\mathbf{r} + \lfloor q/2 \rceil \mathbf{p} - S^T A^T \mathbf{r} \pmod{q} \\ &\equiv (AS)^T\mathbf{r} + E^T\mathbf{r} + \lfloor q/2 \rceil \mathbf{p} - S^T A^T \mathbf{r} \pmod{q} \\ &\equiv E^T\mathbf{r} + \lfloor q/2 \rceil \mathbf{p} \pmod{q} \end{aligned}$$

as long as E is small enough that the coordinates of $E^T\mathbf{r}$ have magnitude less than $q/4$, $\mathbf{p}'$ will be equal to $\mathbf{p}$.

We can replace the vectors with polynomials to get the Ring LWE (RLWE) problem.

Given a polynomial $t(x)$ of the form

$$t(x) \equiv a(x)s(x) + e(x) \pmod{x^n + 1} \pmod{q}, \tag{3}$$

where

- $a(x)$ is a public polynomial in $R = \{\text{polynomials in } x \text{ of degree} \leq n\}$
- $e(x)$ is an “error polynomial” with small coefficients

find the secret polynomial $s(x)$ in R.

Basic RLWE key generation:

Bernie's private key is a polynomial $s(x)$ in R and his public key is $(a(x), t(x))$, where

$$t(x) \equiv a(x)s(x) + e(x) \pmod{x^n + 1} \pmod{q} \tag{4}$$

for an error polynomial $e(x)$.

Basic RLWE encryption:

$p(x)$ is a plaintext message represented as a polynomial in R with 0's and 1's as coefficients.

Aretha chooses

- a random nonce polynomial $r(x)$ in R
- small random error polynomials $e_1(x)$ and $e_2(x)$.

She computes the ciphertext $(u(x), v(x))$, where

$$\begin{aligned} u(x) &= a(x)r(x) + e_1(x) \text{ MOD } x^n + 1 \text{ MOD } q \\ v(x) &= t(x)r(x) + e_2(x) + \lfloor q/2 \rceil\, p(x) \text{ MOD } x^n + 1 \text{ MOD } q. \end{aligned}$$

Basic RLWE decryption:

Bernie again tests each coefficient to see if it is closer to 0 or to $q/2$ modulo q:

$$p'(x) = \left\lfloor \lfloor q/2 \rceil^{-1} \left(v(x) - s(x)u(x)\right) \right\rceil \operatorname{MOD} 2,$$

where the rounding is done component-wise.

Once again, $p'(x)$ will be equal to $p(x)$ as long as the coefficients of $e(x)r(x)$ have magnitude less than $q/4$.

RLWE is more efficient, but possibly less secure.

$a(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0, \quad s(x) = s_{n-1}x^{n-1} + \ldots + s_1 x + s_0$

$$\text{Let } A = \begin{pmatrix} a_0 & -a_{n-1} & \cdots & -a_2 & -a_1 \\ a_1 & a_0 & \cdots & -a_3 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_0 & -a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \end{pmatrix}, \qquad \mathbf{s} = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-2} \\ s_{n-1} \end{pmatrix}. \qquad (5)$$

$$\text{Then } A\mathbf{s} = \begin{pmatrix} a_0 s_0 - a_{n-1}s_1 - \cdots - a_2 s_{n-2} - a_1 s_{n-1} \\ a_0 s_1 + a_0 s_1 - \cdots - a_3 s_{n-2} - a_2 s_{n-1} \\ \vdots \\ a_{n-2}s_0 + a_{n-3}s_1 + \cdots + a_0 s_{n-2} - a_{n-1}s_{n-1} \\ a_{n-1}s_0 + a_{n-2}s_1 + \cdots + a_1 s_{n-2} + a_0 s_{n-1} \end{pmatrix}$$

corresponds to $a(x)s(x)$ reduced modulo $x^n + 1$.

So RLWE is equivalent to LWE with shorter keys but more structure.

Kyber uses a combination of LWE and RLWE called the Module Learning With Errors (MLWE) problem.

Given a vector $\mathbf{t}(x)$ of k polynomials in R, with the form

$$\mathbf{t}(x) \equiv A(x)\mathbf{s}(x) + \mathbf{e}(x) \pmod{x^n + 1} \pmod{q}, \tag{6}$$

where

- $A(x)$ is a $k \times k$ public matrix of polynomials in R with at least as many rows as columns, and
- $\mathbf{e}(x)$ is a “small error vector” of k polynomials in R drawn from some probability distribution

find the secret vector of k polynomials $\mathbf{s}(x)$.

Basic MLWE key generation:

Bernie

- picks $k \times k$ matrix $A(x)$ with uniform random entries in $R_q = \{\text{polynomials in } R \text{ with coefficients in } [0, q)\}$
- generates $2nk\eta$ random bits
- uses random bits to compute centered binomial coefficients for vectors $\mathbf{s}(x)$ and $\mathbf{e}(x)$ of k polynomials in R
- computes $\mathbf{t}(x) = A(x) \star \mathbf{s}(x) + \mathbf{e}(x)$
- posts public encryption key $(A(x), \mathbf{t}(x))$
- keeps private key $\mathbf{s}(x)$ secret

Example ($n = 4, k = 2, q = 23$)

Bernie first generates the random public matrix

$$A(x) = \begin{pmatrix} 4x^3 + 4x^2 + 10x & 11x^3 + 15x^2 + 10x + 3 \\ 12x^3 + 22x^2 + 4x + 12 & 6x^3 + x + 11 \end{pmatrix}$$

and 16 random bits, which he uses to compute secret vectors

$$\mathbf{s}(x) = \begin{pmatrix} -x^3 + x^2 + 1 \\ -x^2 \end{pmatrix}, \qquad \mathbf{e}(x) = \begin{pmatrix} x^2 - x - 1 \\ -x + 1 \end{pmatrix}.$$

He then uses these to compute the public vector

$$\begin{aligned} \mathbf{t}(x) &= A(x) \star \mathbf{s}(x) + \mathbf{e}(x) \\ &= \begin{pmatrix} 4x^3 + 6x^2 + 20x + 20 \\ 3x^3 + 12x^2 + 19x - 5 \end{pmatrix} \end{aligned}$$

and publishes $(A(x), \mathbf{t}(x))$ as his public key.

Basic MLWE encryption:

Aretha

- encodes plaintext as a polynomial $p(x)$ in R
- looks up Bernie's encryption key $(A(x), \mathbf{t}(x))$
- generates $2nk\eta + 2n\eta$ random bits
- uses random bits to compute centered binomial coefficients for vectors $\mathbf{r}(x)$ and $\mathbf{e}_1(x)$ of k polynomials in R each and one polynomial $e_2(x)$ in R
- Computes ciphertext $\mathbf{u}(x) = A(x)^T \star \mathbf{r}(x) + \mathbf{e}_1(x)$, $v(x) = \mathbf{t}(x)^T \star \mathbf{r}(x) + e_2(x) + \lfloor q/2 \rceil\, p(x)$

Example (continued)

Aretha wants to send the message “hi” to Bernie. For the first letter, Aretha encodes “h” as the number 8, or 1000 in binary, corresponding to

$$\mathbf{p}(x) = 1x^3 + 0x^2 + 0x + 0 = x^3.$$

She generates 20 random bits and uses them to compute

$$\mathbf{r}(x) = \begin{pmatrix} x^3 + x^2 + x + 1 \\ -x^3 \end{pmatrix}, \qquad \mathbf{e_1}(x) = \begin{pmatrix} x^3 + x - 1 \\ -x^3 + x^2 - 1 \end{pmatrix},$$

$e_2(x) = -x^2 + x + 1.$ She then computes

$$\mathbf{u}(x) = A(x)^T \star \mathbf{r}(x) + \mathbf{e}_1(x) = \begin{pmatrix} 7x^3 + 22x^2 + 2x - 15 \\ 4x^3 + x^2 - 13x + 13 \end{pmatrix}$$

$$\begin{aligned} v(x) &= \mathbf{t}(x)^T \star \mathbf{r}(x) + e_2(x) + \lfloor q/2 \rceil \, p(x) \\ &= \left[(9x^3 + 22x^2 + 19x + 9) + (-x^2 + x + 1) + 12x^3 \right] \text{MOD } 23 \\ &= 21x^3 + 21x^2 + 20x + 10 \end{aligned}$$

and sends $(\mathbf{u}(x), v(x))$ to Bernie.

Basic MLWE decryption:

Aretha

$$\rightarrow (\mathbf{u}(x), v(x)) \rightarrow$$

Bernie

Bernie

$$(\mathbf{u}(x), v(x))$$

$$\downarrow \mathbf{s}(x)$$

$$p'(x) = \left\lfloor \lfloor q/2 \rceil^{-1} \left(v(x) - \mathbf{s}(x)^T \star \mathbf{u}(x) \right) \right\rceil \text{ MOD } 2$$

Example (concluded)

Bernie receives the message

$$\mathbf{u}(x) = \begin{pmatrix} 7x^3 + 22x^2 + 2x - 15 \\ 4x^3 + x^2 - 13x + 13 \end{pmatrix}$$
$$v(x) = 21x^3 + 21x^2 + 20x + 10$$

from Aretha. He computes

$$\begin{aligned}
&\left\lfloor \lfloor q/2 \rceil^{-1} \left(v(x) - \mathbf{s}(x)^T \star \mathbf{u}(x) \right) \right\rceil \operatorname{MOD} 2 \\
&= \left\lfloor \frac{1}{12}(-16x^3 + 20x^2 - x + 21) \right\rceil \operatorname{MOD} 2 \\
&= \left\lfloor -\frac{4}{3}x^3 + \frac{5}{3}x^2 - \frac{1}{12}x + \frac{7}{4} \right\rceil \operatorname{MOD} 2 \\
&= \left(-x^3 + 2x^2 + 0x + 2 \right) \operatorname{MOD} 2 = x^3 + 0x^2 + 0x + 0,
\end{aligned}$$

yielding the correct plaintext.

The developers of Kyber discovered some tricks that lead to even greater efficiency.

These fall into three main categories:

- using hash function and extendable output functions to generate keys from smaller random numbers,
- using the “number-theoretic transform” (NTT) to speed up polynomial modular multiplication,
- using compression functions to discard some bits in the ciphertext which are unlikely to affect the decryption.

And, of course, they used cryptographically-sized parameters.

	n	k	q	η	$\delta =$ failure probability
Kyber512	256	2	3329	3/2*	2^{-139}
Kyber768	256	3	3329	2	2^{-164}
Kyber1024	256	4	3329	2	2^{-174}

*Kyber512 uses $\eta = 3$ for **s**, **e**, and **r**, and $\eta = 2$ for $\mathbf{e}_1$ and e_2

Table: Parameter sets for Kyber

The best known attack on Kyber is currently the "primal attack" on generic LWE.

The primal attack on LWE starts with the observation that since

$$A\mathbf{s} + \mathbf{e} - \mathbf{t} \equiv \mathbf{0} \pmod{q},$$

we can consider matrices

$$\mathbf{s}'' = \left(\mathbf{s}^T \middle| \mathbf{e}^T \middle| 1 \middle| \mathbf{s}'^T \right), \qquad M = \begin{pmatrix} A^T \\ \hline I \\ \hline -\mathbf{t}^T \\ \hline qI \end{pmatrix}$$

such that $\mathbf{s}''M = \mathbf{0}$.

In other words, $\mathbf{s}''$ is in the left kernel of M, which is constructed from public information. We want to find a short vector in that left kernel.

The first step is to find a basis for the left kernel.

Consider the LWE cryptosystem with $\ell = 1$ and a public key

$$A = \begin{pmatrix} 18 & 10 \\ 16 & 4 \end{pmatrix}, \qquad T = \begin{pmatrix} 15 \\ 0 \end{pmatrix}.$$

If Eve wants to recover S, she can first set up the matrix

$$M' = \left(\begin{array}{rr|ccccccc} 18 & 16 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 10 & 4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \cline{1-2} \cdashline{3-9} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \cline{1-2} \cdashline{3-9} -15 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \cline{1-2} 23 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 23 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Example of finding a basis, continued

Putting this in integer echelon form gives

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -18 & -16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -10 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -3 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -23 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -23 & -15 & 0 \end{pmatrix}$$

A basis for the integer left kernel is

$\{(1\ 0\ -18\ -16\ 0\ 0\ 0), (0\ 1\ -10\ -4\ 0\ 0\ 0)$, $(0\ 0\ 1\ 0\ \text{-}3\ \text{-}2\ 0)$,
$(0\ 0\ 0\ \text{-}23\ 0\ 0\ 1), (0\ 0\ 0\ 0\ \text{-}23\ \text{-}15\ 0)\}$.

We can then use a lattice reduction algorithm to find a short vector of the correct form.

Performing LLL on the matrix composed of the integer left kernel from the previous example yields

$$\begin{pmatrix} 1 & 0 & -18 & -16 & 0 & 0 & 0 \\ 0 & 1 & -10 & -4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -3 & -2 & 0 \\ 0 & 0 & 0 & -23 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -23 & -15 & 0 \end{pmatrix} \xrightarrow{LLL} \begin{pmatrix} 0 & 0 & 1 & 0 & -3 & -2 & 0 \\ -2 & 2 & 1 & 1 & -1 & 0 & 1 \\ 1 & 2 & 0 & -1 & 1 & -1 & -1 \\ 1 & 1 & 2 & 3 & 2 & 0 & -1 \\ -1 & -1 & 5 & -3 & 0 & 1 & 1 \end{pmatrix}$$

Looking for a short vector with a 1 in the correct position for $\mathbf{s}''$ produces

$$\begin{pmatrix} 1 \; 2 \; 0 \; -1 \; 1 \; -1 \; -1 \end{pmatrix},$$

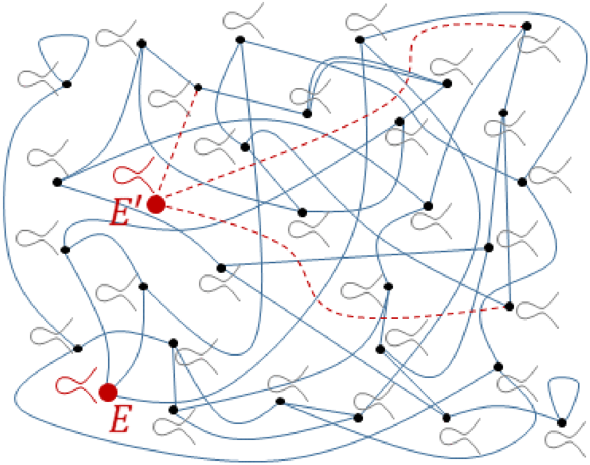
corresponding to $\mathbf{s} = \begin{pmatrix} 1 \; 2 \end{pmatrix}^T$ and $\mathbf{e} = \begin{pmatrix} 0 \; -1 \end{pmatrix}^T$, which is in fact correct.

What kinds of PQC can you teach to undergraduates?

- Lattice-based: Alkaline (a toy version of Kyber: `https://github.com/joshuarbholden/alkaline`) or NTRU
- Code-based: Classic McEliece
- Hash-based: Merkle signature scheme
- Multivariable: Oil and Vinegar
- Elliptic curve isogenies: Charles-Goren-Lauter hash function

My *Resource Guide for Teaching Post-Quantum Cryptography* is at `https://arxiv.org/abs/2207.00558` (to appear in *Cryptologia*).

Thanks for listening!



[Wouter Castryck]