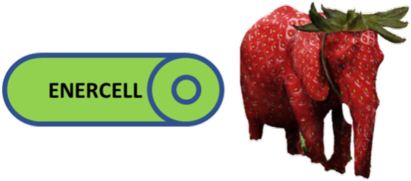
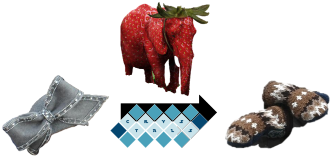




Dilithium/2 = Lithium?

Post-quantum signatures for undergraduate classes

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), Kaskaskia, 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 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)

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

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)

My version of CRYSTALS-Kyber, called “Alkaline”, appeared in this seminar and in “Alkaline: A Simplified Post-Quantum Encryption Algorithm for Classroom Use.” *PRIMUS* 34 (1): 98–122.

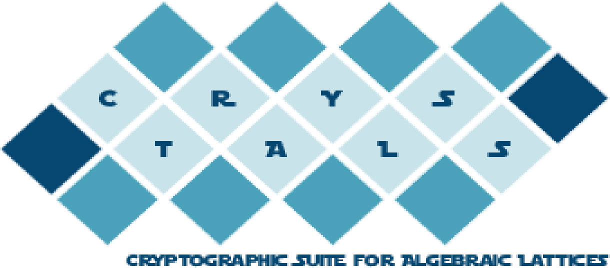
NIST has so far selected four submissions for standardization.

- 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)

Today I will talk about CRYSTALS-Dilithium, the primary digital signature algorithm approved so far.

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

Given a vector **t** of the form

$$\mathbf{t} \equiv A\mathbf{s_1} + \mathbf{s_2} \pmod{q}, \tag{1}$$

where

- A is a public matrix with at least as many rows as columns
- $\mathbf{s_2}$ is a “small error vector” drawn from some probability distribution

find the secret vectors $\mathbf{s_1}$ and/or $\mathbf{s_2}$.

Basic LWE key generation:

Aretha's private key is a pair of "small" (coefficients) matrices S_1 and S_2 and her public key is (A, T), where

$$T \equiv AS_1 + S_2 \pmod{q} \tag{2}$$

with A, T, and S_2 having size $k \times \ell$, and S_1 having size $\ell \times \ell$.

Dilithium signatures are based on LWE and “Fiat-Shamir with Aborts” (Lyubashevsky, ’09 and ’12).

To sign a message M, Aretha:

1. chooses random “small” (coefficients) nonce vectors $\mathbf{y_1}$ and $\mathbf{y_2}$
2. calculates $\mathbf{c} = \mathrm{H}\big(M \parallel (A\mathbf{y_1} + \mathbf{y_2})\,\mathrm{MOD}\ q\big)$ for some vector-valued hash function H which outputs “small” vectors [Fiat-Shamir]
3. calculates $\mathbf{z_1} = \mathbf{y_1} + S_1\mathbf{c}$ and $\mathbf{z_2} = \mathbf{y_2} + S_2\mathbf{c}$
4. if $\mathbf{z_1}$ or $\mathbf{z_2}$ is “too large” then go back to Step 1
5. Otherwise, the signature is $\big(\mathbf{z_1}, \mathbf{z_2}, \mathbf{c}\big)$

(The symbol $\parallel$ is string concatenation. MOD is the “coder’s mod”.)

Dilithium signatures are based on LWE and “Fiat-Shamir with Aborts” (Lyubashevsky, ’09 and ’12).

To sign a message M, Aretha:

1. chooses random “small” (coefficients) nonce vectors $\mathbf{y_1}$ and $\mathbf{y_2}$
2. calculates $\mathbf{c} = \mathrm{H}\big(M \parallel (A\mathbf{y_1} + \mathbf{y_2}) \,\mathrm{MOD}\, q\big)$ for some vector-valued hash function H which outputs “small” vectors [Fiat-Shamir]
3. calculates $\mathbf{z_1} = \mathbf{y_1} + S_1\mathbf{c}$ and $\mathbf{z_2} = \mathbf{y_2} + S_2\mathbf{c}$
4. if $\mathbf{z_1}$ or $\mathbf{z_2}$ is “too large” then go back to Step 1
5. Otherwise, the signature is $(\mathbf{z_1}, \mathbf{z_2}, \mathbf{c})$

Bernie accepts the signature if:

1. $\mathbf{z_1}$ and $\mathbf{z_2}$ are not “too large”, and
2. $\mathbf{c} = \mathrm{H}\big(M \parallel (A\mathbf{z_1} + \mathbf{z_2} - T\mathbf{c}) \,\mathrm{MOD}\, q\big)$

Dilithium signatures are based on LWE and "Fiat-Shamir with Aborts" (Lyubashevsky, '09 and '12).

To sign a message M, Aretha:

1. chooses random "small" (coefficients) nonce vectors $\mathbf{y_1}$ and $\mathbf{y_2}$
2. calculates $\mathbf{c} = \mathrm{H}\big(M \parallel (A\mathbf{y_1} + \mathbf{y_2})\,\mathrm{MOD}\ q\big)$ for some vector-valued hash function H which outputs "small" vectors [Fiat-Shamir]
3. calculates $\mathbf{z_1} = \mathbf{y_1} + S_1\mathbf{c}$ and $\mathbf{z_2} = \mathbf{y_2} + S_2\mathbf{c}$
4. if $\mathbf{z_1}$ or $\mathbf{z_2}$ is "too large" then go back to Step 1
5. Otherwise, the signature is $(\mathbf{z_1}, \mathbf{z_2}, \mathbf{c})$

What's up with Step 4?

Dilithium signatures are based on LWE and “Fiat-Shamir with Aborts” (Lyubashevsky, ’09 and ’12).

To sign a message M, Aretha:

1. chooses random “small” (coefficients) nonce vectors $\mathbf{y_1}$ and $\mathbf{y_2}$
2. calculates $\mathbf{c} = \mathrm{H}\big(M \,\|\, (A\mathbf{y_1} + \mathbf{y_2})\,\mathrm{MOD}\;q\big)$ for some vector-valued hash function H which outputs “small” vectors [Fiat-Shamir]
3. calculates $\mathbf{z_1} = \mathbf{y_1} + S_1\mathbf{c}$ and $\mathbf{z_2} = \mathbf{y_2} + S_2\mathbf{c}$
4. if $\mathbf{z_1}$ or $\mathbf{z_2}$ is “too large” then go back to Step 1
5. Otherwise, the signature is $(\mathbf{z_1}, \mathbf{z_2}, \mathbf{c})$

What’s up with Step 4?

- This is the “abort”, a.k.a “rejection sampling”.
- The goal is to make $\mathbf{z_1}$ and $\mathbf{z_2}$ look uniformly distributed, so that no statistical information about S_1 and S_2 leaks out.

Dilithium makes several changes for efficiency.
The one we want is taken from Bai and Galbraith ('14).

To sign a message M, Aretha:

1. chooses a random “small” (coefficients) nonce vector $\mathbf{y}$
2. calculates $\mathbf{c} = \mathrm{H}\big(M \,\|\, \mathrm{HighBits}(A\mathbf{y} \bmod q)\big)$ for some vector-valued hash function H which outputs “small” vectors
3. calculates $\mathbf{z} = \big(\mathbf{y} + S_1\mathbf{c}\big) \bmod q$ [no $\mathbf{z_2}$!]
4. if $\mathbf{z}$ is “too large” or $\mathrm{LowBits}\big((A\mathbf{y} - S_2\mathbf{c}) \bmod q\big)$ is “too large” then go back to Step 1
5. Otherwise, the signature is $(\mathbf{z}, \mathbf{c})$ [smaller than before!]

Dilithium makes several changes for efficiency.

The one we want is taken from Bai and Galbraith ('14).

To sign a message M, Aretha:

1. chooses a random "small" (coefficients) nonce vector $\mathbf{y}$
2. calculates $\mathbf{c} = \mathrm{H}\big(M \parallel \mathrm{HighBits}\big(A\mathbf{y} \text{ MOD } q\big)\big)$ for some vector-valued hash function H which outputs "small" vectors
3. calculates $\mathbf{z} = \big(\mathbf{y} + S_1\mathbf{c}\big) \text{ MOD } q$ [no $\mathbf{z_2}$!]
4. if $\mathbf{z}$ is "too large" or $\mathrm{LowBits}\big(\big(A\mathbf{y} - S_2\mathbf{c}\big) \text{ MOD } q\big)$ is "too large" then go back to Step 1
5. Otherwise, the signature is $\big(\mathbf{z}, \mathbf{c}\big)$ [smaller than before!]

Bernie accepts the signature if:

1. $\mathbf{z}$ is not "too large", and
2. $\mathbf{c} = \mathrm{H}\big(M \parallel \mathrm{HighBits}\big(\big(A\mathbf{z} - T\mathbf{c}\big) \text{ MOD } q\big)\big)$

Dilithium makes several changes for efficiency. The one we want is taken from Bai and Galbraith ('14).

To sign a message M, Aretha:

1. chooses a random "small" (coefficients) nonce vector $\mathbf{y}$
2. calculates $\mathbf{c} = \mathrm{H}\big(M \,\|\, \mathrm{HighBits}\big(A\mathbf{y} \bmod q\big)\big)$ for some vector-valued hash function H which outputs "small" vectors
3. calculates $\mathbf{z} = \big(\mathbf{y} + S_1\mathbf{c}\big) \bmod q$ [no $\mathbf{z_2}$!]
4. if $\mathbf{z}$ is "too large" or $\mathrm{LowBits}\big(\big(A\mathbf{y} - S_2\mathbf{c}\big) \bmod q\big)$ is "too large" then go back to Step 1
5. Otherwise, the signature is $\big(\mathbf{z}, \mathbf{c}\big)$ [smaller than before!]

Now what's up with Step 4?

Dilithium makes several changes for efficiency. The one we want is taken from Bai and Galbraith ('14).

To sign a message M, Aretha:

1. chooses a random "small" (coefficients) nonce vector $\mathbf{y}$
2. calculates $\mathbf{c} = \mathrm{H}(M \parallel \mathrm{HighBits}(A\mathbf{y}\ \mathrm{MOD}\ q))$ for some vector-valued hash function H which outputs "small" vectors
3. calculates $\mathbf{z} = (\mathbf{y} + S_1\mathbf{c})\ \mathrm{MOD}\ q$ [no $\mathbf{z_2}$!]
4. if $\mathbf{z}$ is "too large" or $\mathrm{LowBits}((A\mathbf{y} - S_2\mathbf{c})\ \mathrm{MOD}\ q)$ is "too large" then go back to Step 1
5. Otherwise, the signature is $(\mathbf{z}, \mathbf{c})$ [smaller than before!]

Now what's up with Step 4?

- $A\mathbf{y}$ and $A\mathbf{z} - T\mathbf{c}$ now differ by $S_2\mathbf{c}$
- However, S_2 and $\mathbf{c}$ are small, so if $\mathrm{LowBits}(A\mathbf{y} - S_2\mathbf{c})$ is small, then the difference won't overflow into the "HighBits" parts of $A\mathbf{y}$ and $A\mathbf{z} - T\mathbf{c}$

How could Frank the Forger attack this?

1. He could try to recover S_1 and/or S_2 from A and $\left(AS_1 + S_2\right)$ MOD q. This is LWE.

How could Frank the Forger attack this?

1. He could try to recover S_1 and/or S_2 from A and $\left(AS_1 + S_2\right)$ MOD q. This is LWE.
2. He could pick a random **y** and calculate **c**, then try to find a **z** which Bernie will accept. This requires finding a preimage of the hash function.

How could Frank the Forger attack this?

1. He could try to recover S_1 and/or S_2 from A and $(AS_1 + S_2) \text{ MOD } q$. This is LWE.
2. He could pick a random $\mathbf{y}$ and calculate $\mathbf{c}$, then try to find a $\mathbf{z}$ which Bernie will accept. This requires finding a preimage of the hash function.
3. He could pick a random $\hat{\mathbf{z}}$ and $\hat{\mathbf{c}}$, calculate

$$\mathbf{c} = \mathrm{H}\big(M \parallel \mathrm{HighBits}\big((A\hat{\mathbf{z}} - T\hat{\mathbf{c}}) \text{ MOD } q\big)\big),$$

and try to find $\mathbf{z}$ such that

$$\mathbf{c} = \mathrm{H}\big(M \parallel \mathrm{HighBits}\big((A\mathbf{z} - T\mathbf{c}) \text{ MOD } q\big)\big).$$

This requires finding a second preimage of the hash function.

How could Frank the Forger attack this?

1. He could try to recover S_1 and/or S_2 from A and $(AS_1 + S_2)\,\mathrm{MOD}\,q$. This is LWE.
2. He could pick a random $\mathbf{y}$ and calculate $\mathbf{c}$, then try to find a $\mathbf{z}$ which Bernie will accept. This requires finding a preimage of the hash function.
3. He could pick a random $\hat{\mathbf{z}}$ and $\hat{\mathbf{c}}$, calculate

   $$\mathbf{c} = \mathrm{H}\big(M \parallel \mathrm{HighBits}\big((A\hat{\mathbf{z}} - T\hat{\mathbf{c}})\,\mathrm{MOD}\,q\big)\big),$$

   and try to find $\mathbf{z}$ such that

   $$\mathbf{c} = \mathrm{H}\big(M \parallel \mathrm{HighBits}\big((A\mathbf{z} - T\mathbf{c})\,\mathrm{MOD}\,q\big)\big).$$

   This requires finding a second preimage of the hash function.
4. He could pick a random $\hat{\mathbf{z}}$ and $\hat{\mathbf{c}}$, calculate

   $$\mathbf{c} = \mathrm{H}\big(M \parallel \mathrm{HighBits}\big((A\hat{\mathbf{z}} - T\hat{\mathbf{c}})\,\mathrm{MOD}\,q\big)\big),$$

   and try to find $\mathbf{z}$ such that $A\mathbf{z} - T\mathbf{c} \approx A\hat{\mathbf{z}} - T\hat{\mathbf{c}}$. This is as hard as SIS.

I call the above version Lithium-LA (Linear Algebra).

Example ($k = 4$, $\ell = 4$, $q = 41$): Aretha first generates the random matrices

$$A = \begin{pmatrix} 3 & 22 & 21 & 19 \\ 19 & 21 & 4 & 1 \\ 6 & 7 & 0 & 2 \\ 1 & 10 & 22 & 0 \end{pmatrix}, \qquad S_1 = \begin{pmatrix} -1 & 0 & -1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & -1 & 0 & 1 \\ -1 & 0 & 0 & 0 \end{pmatrix},$$

$$S_2 = \begin{pmatrix} 0 & -1 & -1 & 0 \\ 1 & -1 & 0 & -1 \\ 0 & -1 & 1 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix}.$$

She then uses these to compute the public matrix

$$T = \big(AS_1 + S_2\big) \operatorname{MOD} 41 = \begin{pmatrix} 27 & 9 & 36 & 22 \\ 24 & 19 & 14 & 21 \\ 21 & 37 & 33 & 23 \\ 3 & 37 & 33 & 30 \end{pmatrix}$$

and publishes $\big(A, T\big)$ as her public verification key.

Lithium-LA example (continued)

We have the further "smallness" parameters $\gamma_1 = 16$, $\gamma_2 = 10$, $\beta = 1$.
Aretha wants to send the message "hi!" to Bernie. She computes a random vector

$$\mathbf{y} = \begin{pmatrix} -3 & 0 & -1 & 14 \end{pmatrix}^T \qquad \text{and} \qquad A\mathbf{y} \,\text{MOD}\, 41 = \begin{pmatrix} 16 & 20 & 26 & 4 \end{pmatrix}^T.$$

$\begin{pmatrix} 16 & 20 & 26 & 4 \end{pmatrix}^T = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix}^T \cdot 2\gamma_2 + \begin{pmatrix} -4 & 0 & 6 & 4 \end{pmatrix}^T$, so $\mathbf{w_1} = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix}^T$.

In Sagemath, $M \parallel \mathbf{w_1} =$ `"hi![1]\n[1]\n[1]\n[0]"`
and our hash function (to be explained shortly) produces $\mathbf{c} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$.

Aretha continues calculating

$$\mathbf{z} = (\mathbf{y} + S_1\mathbf{c})\,\text{MOD}\, q = \begin{pmatrix} 37 & 0 & 0 & 13 \end{pmatrix}^T \equiv \begin{pmatrix} -4 & 0 & 0 & 13 \end{pmatrix}^T \mod 41$$

The coefficients are smaller than $\gamma_1 - \beta$, so the first check passes.

Lithium-LA example (continued)

We have the further "smallness" parameters $\gamma_1 = 16$, $\gamma_2 = 10$, $\beta = 1$.
Aretha wants to send the message "hi!" to Bernie. She computes a random vector

$$\mathbf{y} = \begin{pmatrix} -3 \; 0 \; -1 \; 14 \end{pmatrix}^T \qquad \text{and} \qquad A\mathbf{y} \,\mathrm{MOD}\, 41 = \begin{pmatrix} 16 \; 20 \; 26 \; 4 \end{pmatrix}^T.$$

$\begin{pmatrix} 16 \; 20 \; 26 \; 4 \end{pmatrix}^T = \begin{pmatrix} 1 \; 1 \; 1 \; 0 \end{pmatrix}^T \cdot 2\gamma_2 + \begin{pmatrix} -4 \; 0 \; 6 \; 4 \end{pmatrix}^T$, so $\mathbf{w_1} = \begin{pmatrix} 1 \; 1 \; 1 \; 0 \end{pmatrix}^T$.

In Sagemath, $M \parallel \mathbf{w_1} =$ `"hi![1]\n[1]\n[1]\n[0]"`
and our hash function (to be explained shortly) produces $\mathbf{c} = \begin{pmatrix} 1 \; 0 \; 0 \; 0 \end{pmatrix}^T$.

Aretha continues calculating

$$\begin{pmatrix} A\mathbf{y} - S_2\mathbf{c} \end{pmatrix} \mathrm{MOD}\, 41 = \begin{pmatrix} 16 \; 19 \; 26 \; 4 \end{pmatrix}^T = \begin{pmatrix} 1 \; 1 \; 1 \; 0 \end{pmatrix}^T \cdot 2\gamma_2 + \begin{pmatrix} -4 \; 1 \; 6 \; 4 \end{pmatrix}^T.$$

The low-order bits of $A\mathbf{y} - S_2\mathbf{c}$ are smaller than $\gamma_2 - \beta$, so the second check passes.
Aretha sends the message and the signature $(\mathbf{z}, \mathbf{c})$ to Bernie.

Lithium-LA example (concluded)

Bernie receives the message $M = \texttt{"hi!"}$ and the signature

$$\mathbf{z} = (-4\ 0\ 0\ 13)^T$$
$$\mathbf{c} = (1\ 0\ 0\ 0)^T$$

He checks that the coefficients of $\mathbf{z}$ are smaller than $\gamma_1 - \beta$ and calculates

$$\mathrm{H}\big(M \parallel \mathrm{HighBits}\big((A\mathbf{z} - T\mathbf{c})\ \mathrm{MOD}\ 41\big)\big)$$

Since $\mathrm{HighBits}\big((A\mathbf{z} - T\mathbf{c})\ \mathrm{MOD}\ 41\big) = (1\ 1\ 1\ 0)^T$, this gives the correct result:

$$\mathrm{H}\big(M \parallel (1\ 1\ 1\ 0)^T\big) = (1\ 0\ 0\ 0)^T.$$

Bernie accepts the signature.

How do we do the hashing?

Recall that H is a function that takes in arbitrary strings of bits and outputs "small" vectors.

- Specifically, for some fixed τ, it will output $\mathbf{c}$ such that τ of the entries in $\mathbf{c}$ are ± 1 and the rest are zero.

We would also like H to:

a. be fast to compute,

b. distribute hash values evenly,

1. be resistant to finding a preimage M such that $\mathsf{H}(M) = \mathbf{c}$,
2. be resistant to finding a second preimage M' such that $\mathsf{H}(M') = \mathsf{H}(M)$.
3. (Most cryptographic hash functions also make it hard to find any two colliding M_1 and M_2 such that $\mathsf{H}(M_1) = \mathsf{H}(M_2)$.)

How do we do the hashing?

Recall that H is a function that takes in arbitrary strings of bits and outputs “small” vectors.

- Specifically, for some fixed τ, it will output $\mathbf{c}$ such that τ of the entries in $\mathbf{c}$ are ± 1 and the rest are zero.

Lithium, like Dilithium, uses a hash function (possibly JHA-3 once I write it?) to hash a string representing $M \parallel \mathbf{w_1}$ into a series of bytes twice the length of $\mathbf{c}$. It then applies the following algorithm to construct $\mathbf{c}$:

```
1. $\mathbf{c} = (000 \cdots 0)^T$
2. for $i$ from length$(\mathbf{c}) - \tau$ to length$(\mathbf{c}) - 1$ do
    2.1 $j = \mathrm{H}(M \parallel \mathbf{w_1})_i \ \mathrm{MOD}(i+1)$ [the latest standard has a better way]
    2.2 $s = \mathrm{H}(M \parallel \mathbf{w_1})_{i+\mathrm{length}(c)} \ \mathrm{MOD}\, 2$
    2.3 $\mathbf{c}_i = \mathbf{c}_j$
    2.4 $\mathbf{c}_j = (-1)^s$
3. return $\mathbf{c}$
```

This is based on the “inside-out Fisher-Yates shuffle”.

What about those “smallness” parameters?

- τ is the number of nonzero entries in **c**, giving **c** an “entropy” of $\ln \binom{\ell}{\tau} + \tau$. We would like that entropy to be between $\ell/2$ and ℓ.

What about those “smallness” parameters?

- τ is the number of nonzero entries in **c**, giving **c** an “entropy” of $\ln \binom{\ell}{\tau} + \tau$. We would like that entropy to be between $\ell/2$ and ℓ.
- η is the largest absolute value of coefficients in S_1 and S_2.
- $\beta = \tau \cdot \eta$ is the largest absolute value of the coefficients in $S_1\mathbf{c}$ and $S_2\mathbf{c}$.

What about those “smallness” parameters?

- τ is the number of nonzero entries in **c**, giving **c** an “entropy” of $\ln \binom{\ell}{\tau} + \tau$. We would like that entropy to be between $\ell/2$ and ℓ.
- η is the largest absolute value of coefficients in S_1 and S_2.
- $\beta = \tau \cdot \eta$ is the largest absolute value of the coefficients in $S_1\mathbf{c}$ and $S_2\mathbf{c}$.
- γ_1 is the largest absolute value of coefficients in **y**.
- γ_2 is the largest absolute value of the “low-order bits” of $A\mathbf{y}$, etc.

What about those “smallness” parameters?

- τ is the number of nonzero entries in **c**, giving **c** an “entropy” of $\ln \binom{\ell}{\tau} + \tau$. We would like that entropy to be between $\ell/2$ and ℓ.
- η is the largest absolute value of coefficients in S_1 and S_2.
- $\beta = \tau \cdot \eta$ is the largest absolute value of the coefficients in $S_1\mathbf{c}$ and $S_2\mathbf{c}$.
- γ_1 is the largest absolute value of coefficients in **y**.
- γ_2 is the largest absolute value of the “low-order bits” of $A\mathbf{y}$, etc.
- Thus $\gamma_1 - \beta$ is the largest allowable absolute value in **z**
- and $\gamma_2 - \beta$ is the largest allowable absolute value in $A\mathbf{y} - S_2\mathbf{c}$.

Of course, the choice of parameters involves tradeoffs.

- If γ_1 or γ_2 is too small then signature will leak information about the secret key.
- If γ_1 or γ_2 is too large then the signature will be easy to forge.
- Also, if γ_1 or γ_2 is too large then it will take many tries to generate a successful signature.

The probability that the signature passes both tests is approximately

$$\left(\frac{2(\gamma_1 - \beta) - 1}{2\gamma_1 - 1}\right)^{n \cdot \ell} \left(\frac{2(\gamma_2 - \beta) - 1}{2\gamma_2}\right)^{n \cdot k}.$$

The CRYSTALS team aimed for this to be around 0.25, thus requiring about 4 tries on average to generate a signature.

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

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

$$t(x) \equiv a(x)s_1(x) + s_2(x) \pmod{x^n + 1} \pmod{q}, \tag{3}$$

where

- $a(x)$ is a public polynomial in $R = \{\text{polynomials in } x \text{ of degree} \leq n\}$
- $s_2(x)$ is an “error polynomial” with small coefficients

find the secret polynomials $s_1(x)$ and/or $s_2(x)$ in R.

RLWE is more efficient, but possibly less secure.

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

$$\text{Let } A = \begin{pmatrix} a_0 & -a_{n-1} & \cdots & -a_2 & -a_1 \\ a_1 & a_0 & \cdots & -a_3 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_0 & -a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \end{pmatrix}, \qquad \mathbf{s} = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-2} \\ s_{n-1} \end{pmatrix}. \tag{4}$$

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

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

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

Dilithium 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

$$\mathbf{t}(x) \equiv A(x)\mathbf{s_1}(x) + \mathbf{s_2}(x) \pmod{x^n + 1} \pmod{q}, \tag{5}$$

where

- $A(x)$ is a $k \times k$ public matrix of polynomials in R with at least as many rows as columns, and
- $\mathbf{s_2}(x)$ is a "small error vector" of k polynomials in R drawn from some probability distribution

find the secret vectors of k polynomials $\mathbf{s_1}(x)$ and/or $\mathbf{s_2}(x)$.

The CRYSTALS team discovered some tricks that lead to even greater efficiency.

These fall into two 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 $\mathbf{t}(x)$ and replacing them with a smaller “hint” **h**.

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

	n	(k, ℓ)	q	τ	η	expected repetitions
Level 2	256	(4,4)	8380417	39	2	4.25
Level 3	256	(6,5)	8380417	49	4	5.5
Level 4	256	(8,7)	8380417	60	2	3.85

Table: Parameter sets for Dilithium

For Lithium, I currently plan to have the following parameters:

	n	(k,ℓ)	q	τ	η	expected repetitions
Lithium-LA	1	(4,4)	41	1	1	2.??
Lithium-ALG	4	(1,1)	41	1	1	???
Lithium-AA	4	(2,2)	41	1	1	2.??

Table: Parameter sets for Dilithium

The best known attack on Dilithium 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

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

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

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

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 undergraduate classes would this be appropriate for?

- Linear Algebra: Alkaline-LA, Lithium-LA
- Abstract Algebra: Alkaline-ALG or -AA, Lithium-ALG or -AA
- Second course in Algebra: Alkaline-AA, Lithium-AA
- Probability: The Fisher-Yates shuffle
- Data Structures: Hash functions into “unusual” domains
- Cryptography and/or student research project: any of these!

My *Resource Guide for Teaching Post-Quantum Cryptography* is at `https://arxiv.org/abs/2207.00558` (*Cryptologia*, VOL. 47, NO. 5, 459–465.)

Thanks for listening!