



CRYSTALS – Kyber and Dilithium

Peter Schwabe

peter@cryptojedi.org

<https://cryptojedi.org>

February 7, 2018



5 building blocks for a “secure channel”

Symmetric crypto

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)



5 building blocks for a “secure channel”

Symmetric crypto

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)

Asymmetric crypto

- Key agreement / public-key encryption (e.g., RSA, Diffie-Hellman, ECDH)
- Signatures (e.g., RSA, DSA, ECDSA, EdDSA)



5 building blocks for a “secure channel”

Symmetric crypto

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)

Asymmetric crypto

- Key agreement / public-key encryption (e.g., RSA, Diffie-Hellman, ECDH)
- Signatures (e.g., RSA, DSA, ECDSA, EdDSA)

The asymmetric monoculture

- All widely deployed asymmetric crypto relies on
 - the **hardness of factoring**, or
 - the **hardness of (elliptic-curve) discrete logarithms**



Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*

Peter W. Shor[†]

Abstract

A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, e.g., the number of digits of the integer to be factored.

Definition

Post-quantum crypto is asymmetric crypto that's not based on factoring or DLP.



Definition

Post-quantum crypto is (asymmetric) crypto that resists attacks using classical *and quantum* computers.



Definition

Post-quantum crypto is (asymmetric) crypto that resists attacks using classical *and quantum* computers.

5 main directions

- Lattice-based crypto (PKE and Sigs)
- Code-based crypto (mainly PKE)
- Multivariate-based crypto (mainly Sigs)
- Hash-based signatures (only Sigs)
- Isogeny-based crypto (so far, mainly PKE)



Definition

Post-quantum crypto is (asymmetric) crypto that resists attacks using classical *and quantum* computers.


5 main directions

- **Lattice-based crypto** (PKE and Sigs)
- Code-based crypto (mainly PKE)
- Multivariate-based crypto (mainly Sigs)
- Hash-based signatures (only Sigs)
- Isogeny-based crypto (so far, mainly PKE)



The NIST competition

Count of Problem Category	Column Labels		
Row Labels	Key Exchange	Signature	Grand Total
?	1		1
Braids	1	1	2
Chebychev	1		1
Codes	19	5	24
Finite Automata	1	1	2
Hash		4	4
Hypercomplex Numbers	1		1
Isogeny	1		1
Lattice	24	4	28
Mult. Var	6	7	13
Rand. walk	1		1
RSA	1	1	2
Grand Total	57	23	80



Overview tweeted by Jacob Alperin-Sheriff on Dec 4, 2017.

Status today

- 69 submissions accepted as “complete and proper”
- Several already broken
- 3 withdrawn



- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Given “noise distribution” χ
- Given samples $\mathbf{A}\mathbf{s} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$



- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Given “noise distribution” χ
- Given samples $\mathbf{A}\mathbf{s} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$
- Search version: find \mathbf{s}
- Decision version: distinguish from uniform random



Short integer solution (SIS)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Find nonzero $\mathbf{x} \in \mathbb{Z}^\ell$, s.t.:
 - $\mathbf{Ax} = \mathbf{0} \in \mathbb{Z}_q^k$
 - $\|\mathbf{x}\| < \beta$



Short integer solution (SIS)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Find nonzero $\mathbf{x} \in \mathbb{Z}^\ell$, s.t.:
 - $\mathbf{Ax} = \mathbf{0} \in \mathbb{Z}_q^k$
 - $\|\mathbf{x}\| < \beta$
- Needs $\beta < q$, otherwise trivial



- Problem with LWE/SIS-based cryptosystems: public-key size
- Idea to solve this: allow structured matrix \mathbf{A} :
 - Ring-LWE: work in $\mathbb{Z}_q[X]/(X^n + 1)$; n a power of 2, q typically prime
 - NTRU: work in $\mathbb{Z}_q[X]/(X^n - 1)$; n prime, q a power of 2
 - NTRU Prime: work in $\mathbb{Z}_q[X]/(X^n - X - 1)$; q prime, n prime



- Problem with LWE/SIS-based cryptosystems: public-key size
- Idea to solve this: allow structured matrix \mathbf{A} :
 - Ring-LWE: work in $\mathbb{Z}_q[X]/(X^n + 1)$; n a power of 2, q typically prime
 - NTRU: work in $\mathbb{Z}_q[X]/(X^n - 1)$; n prime, q a power of 2
 - NTRU Prime: work in $\mathbb{Z}_q[X]/(X^n - X - 1)$; q prime, n prime
- Perform arithmetic on polynomials instead of vectors/matrices
- Particularly efficient $\mathbb{Z}_q[X]/(X^n + 1)$, with $n = 2^m$ and $2n \mid (q - 1)$
- Reason: efficient NTT-based multiplication:

$$f \cdot g = \text{NTT}^{-1}(\text{NTT}(f) \circ \text{NTT}(g))$$

- Problem with LWE/SIS-based cryptosystems: public-key size
- Idea to solve this: allow structured matrix \mathbf{A} :
 - Ring-LWE: work in $\mathbb{Z}_q[X]/(X^n + 1)$; n a power of 2, q typically prime
 - NTRU: work in $\mathbb{Z}_q[X]/(X^n - 1)$; n prime, q a power of 2
 - NTRU Prime: work in $\mathbb{Z}_q[X]/(X^n - X - 1)$; q prime, n prime
- Perform arithmetic on polynomials instead of vectors/matrices
- Particularly efficient $\mathbb{Z}_q[X]/(X^n + 1)$, with $n = 2^m$ and $2n \mid (q - 1)$
- Reason: efficient NTT-based multiplication:

$$f \cdot g = \text{NTT}^{-1}(\text{NTT}(f) \circ \text{NTT}(g))$$

- Problem with these highly structured instances of LWE/SIS:
 - Scaling security levels via n : requires re-optimizing code
 - Strong structure in LWE instances may enable attacks

- In CRYSTALS: use matrices and vectors of small dimension $k \times \ell$ over $\mathbb{Z}_q[X]/(X^{256} + 1)$
- Scale security levels by varying k :

```
void polyvec_ntt(polyvec *r) {  
    int i;  
    for(i=0;i<KYBER_K;i++)  
        poly_ntt(&r->vec[i]);  
}
```



CRYSTALS – use module lattices

- In CRYSTALS: use matrices and vectors of small dimension $k \times \ell$ over $\mathbb{Z}_q[X]/(X^{256} + 1)$
- Scale security levels by varying k :

```
void polyvec_ntt(polyvec *r) {  
    int i;  
    for(i=0;i<KYBER_K;i++)  
        poly_ntt(&r->vec[i]);  
}
```

- Breaks some of the structure in LWE/SIS
- Naturally gives us dimension 768



CRYSTALS – use module lattices

- In CRYSTALS: use matrices and vectors of small dimension $k \times \ell$ over $\mathbb{Z}_q[X]/(X^{256} + 1)$
- Scale security levels by varying k :

```
void polyvec_ntt(polyvec *r) {  
    int i;  
    for(i=0;i<KYBER_K;i++)  
        poly_ntt(&r->vec[i]);  
}
```

- Breaks some of the structure in LWE/SIS
- Naturally gives us dimension 768
- Achieves similar performance as Ring-LWE-based systems
- Important for performance: sample uniformly in NTT domain

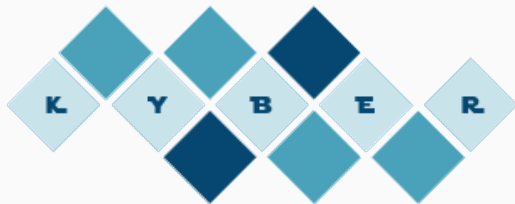


- In CRYSTALS: use matrices and vectors of small dimension $k \times \ell$ over $\mathbb{Z}_q[X]/(X^{256} + 1)$
- Scale security levels by varying k :

```
void polyvec_ntt(polyvec *r) {
    int i;
    for(i=0;i<KYBER_K;i++)
        poly_ntt(&r->vec[i]);
}
```

- Breaks some of the structure in LWE/SIS
- Naturally gives us dimension 768
- Achieves similar performance as Ring-LWE-based systems
- Important for performance: sample uniformly in NTT domain
- For Kyber use $q = 7681$, for Dilithium $q = 8380417$





Kyber: The KEM

joint work with

Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint,
Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé



- Inspired by NewHope, can see it as successor:
 - Against-all-authority approach for generating **A**
 - Centered binomial noise (no discrete Gaussians)
 - Conservative parameters and analysis
 - Easy and efficient to implement



- Inspired by NewHope, can see it as successor:
 - Against-all-authority approach for generating \mathbf{A}
 - Centered binomial noise (no discrete Gaussians)
 - Conservative parameters and analysis
 - Easy and efficient to implement
- Improvements:
 - Module-LWE instead of Ring-LWE
 - CCA-secure instead of CPA-secure (now also adopted by NewHope)



Gen()

$$\mathbf{s}, \mathbf{e} \leftarrow \chi$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$$

$$pk = \mathbf{t}, sk = \mathbf{s}$$

Enc($pk, m \in \{0, 1\}^{256}$)

$$\mathbf{r}, \mathbf{e}_1 \leftarrow \chi$$

$$\mathbf{u} \leftarrow \mathbf{A}^T \mathbf{r} + \mathbf{e}_1$$

$$\mathbf{t}^T \mathbf{r}$$

$$c = \mathbf{u}$$

Dec(sk, c)

$$\mathbf{s}^T \mathbf{u}$$



Kyber.CPAPKE (“noisy ElGamal”)

Gen()

$$\rho \leftarrow \{0, 1\}^{256}$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{s}, \mathbf{e} \leftarrow \chi$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$$

$$pk = (\mathbf{t}, \rho), sk = \mathbf{s}$$

Enc($pk, m \in \{0, 1\}^{256}$)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{r}, \mathbf{e}_1 \leftarrow \chi$$

$$\mathbf{u} \leftarrow \mathbf{A}^T \mathbf{r} + \mathbf{e}_1$$

$$c = \mathbf{u}$$

Dec(sk, c)

$$\mathbf{s}^T \mathbf{u}$$



Kyber.CPAPKE (“noisy ElGamal”)

Gen()

$$\rho \leftarrow \{0, 1\}^{256}$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{s}, \mathbf{e} \leftarrow \chi$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$$

$$pk = (\mathbf{t}, \rho), sk = \mathbf{s}$$

Enc($pk, m \in \{0, 1\}^{256}$)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow \chi$$

$$\mathbf{u} \leftarrow$$

$$\mathbf{v} \leftarrow$$

$$c = (\mathbf{u}, \mathbf{v})$$

$$\mathbf{A}^T \mathbf{r} + \mathbf{e}_1$$

$$\mathbf{t}^T \mathbf{r} + \mathbf{e}_2 + \left\lceil \frac{q}{2} \right\rceil \cdot m$$

Dec(sk, c)

$$m = \text{Compress}(\mathbf{v} - \mathbf{s}^T \mathbf{u}, 1)$$

Gen()

$$\rho \leftarrow \{0, 1\}^{256}$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{s}, \mathbf{e} \leftarrow \chi$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$$

$$pk = (\mathbf{t}, \rho), sk = \mathbf{s}$$

Enc($pk, m \in \{0, 1\}^{256}$)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow \chi$$

$$\mathbf{u} \leftarrow \mathbf{A}^T \mathbf{r} + \mathbf{e}_1$$

$$v \leftarrow \text{Compress}(\mathbf{t}^T \mathbf{r} + \mathbf{e}_2 + \lceil \frac{q}{2} \rceil \cdot m, d_v)$$

$$c = (\mathbf{u}, v)$$

Dec(sk, c)

$$v := \text{Decompress}(v, d_v)$$

$$m = \text{Compress}(v - \mathbf{s}^T \mathbf{u}, 1)$$

Kyber.CPAPKE (“noisy ElGamal”)

Gen()

$$\rho \leftarrow \{0, 1\}^{256}$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{s}, \mathbf{e} \leftarrow \chi$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$$

$$pk = (\mathbf{t}, \rho), sk = \mathbf{s}$$

Dec(sk, c)

$$\mathbf{u} := \text{Decompress}(\mathbf{u}, d_u)$$

$$v := \text{Decompress}(v, d_v)$$

$$m = \text{Compress}(v - \mathbf{s}^T \mathbf{u}, 1)$$

Enc($pk, m \in \{0, 1\}^{256}$)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow \chi$$

$$\mathbf{u} \leftarrow \text{Compress}(\mathbf{A}^T \mathbf{r} + \mathbf{e}_1, d_u)$$

$$v \leftarrow \text{Compress}(\mathbf{t}^T \mathbf{r} + \mathbf{e}_2 + \left\lceil \frac{q}{2} \right\rceil \cdot m, d_v)$$

$$c = (\mathbf{u}, v)$$

Gen()

$$\rho \leftarrow \{0, 1\}^{256}$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{s}, \mathbf{e} \leftarrow \chi$$

$$\mathbf{t} \leftarrow \text{Compress}(\mathbf{A}\mathbf{s} + \mathbf{e}, d_t)$$

$$pk = (\mathbf{t}, \rho), sk = \mathbf{s}$$

Dec(sk, c)

$$\mathbf{u} := \text{Decompress}(\mathbf{u}, d_u)$$

$$v := \text{Decompress}(v, d_v)$$

$$m = \text{Compress}(v - \mathbf{s}^T \mathbf{u}, 1)$$

Enc($pk, m \in \{0, 1\}^{256}$)

$$\mathbf{t} := \text{Decompress}(\mathbf{t}, d_t)$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow \chi$$

$$\mathbf{u} \leftarrow \text{Compress}(\mathbf{A}^T \mathbf{r} + \mathbf{e}_1, d_u)$$

$$v \leftarrow \text{Compress}(\mathbf{t}^T \mathbf{r} + \mathbf{e}_2 + \left\lceil \frac{q}{2} \right\rceil \cdot m, d_v)$$

$$c = (\mathbf{u}, v)$$

- When compressing the public key, v is not an MLWE sample
- $t = \text{Decompress}(\text{Compress}(\mathbf{As} + \mathbf{e}, d_t)), d_t)$ is not uniform
- This was pointed out by Jan Pieter D'Anvers



A point we missed...

- When compressing the public key, v is not an MLWE sample
- $t = \text{Decompress}(\text{Compress}(\mathbf{As} + \mathbf{e}, d_t)), d_t)$ is not uniform
- This was pointed out by Jan Pieter D'Anvers
- Possible fix: re-randomize after decompression
- Not easy/efficient to do with the current compression
- Can simply drop bits
 - Easy and efficient to re-randomize
 - Introduces more “deterministic noise”



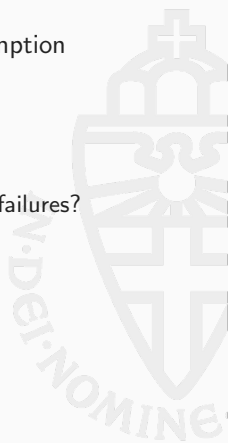
- When compressing the public key, v is not an MLWE sample
- $t = \text{Decompress}(\text{Compress}(\mathbf{As} + \mathbf{e}, d_t)), d_t)$ is not uniform
- This was pointed out by Jan Pieter D'Anvers
- Possible fix: re-randomize after decompression
- Not easy/efficient to do with the current compression
- Can simply drop bits
 - Easy and efficient to re-randomize
 - Introduces more “deterministic noise”
- Doesn't lead to an actual attack
 - Compression of v hides almost all differences
 - On average, 4% of coeffs are different w and w/o compression of t



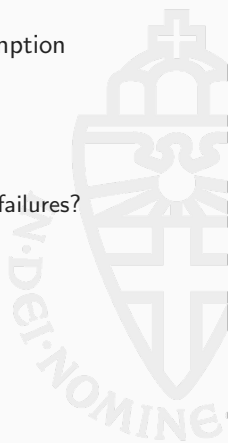
- Tight reduction from MLWE in the ROM (if we didn't compress pk)
- Non-tight reduction in the QROM
- Tight reduction in the QROM with non-standard assumption



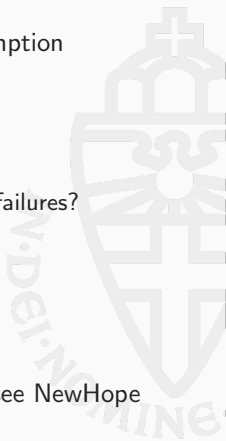
- Tight reduction from MLWE in the ROM (if we didn't compress pk)
- Non-tight reduction in the QROM
- Tight reduction in the QROM with non-standard assumption
- Failure probability of $< 2^{-140}$
- Interesting questions:
 - How much of a problem are a few failures?
 - How much can an attacker exploit Grover to produce failures?



- Tight reduction from MLWE in the ROM (if we didn't compress pk)
- Non-tight reduction in the QROM
- Tight reduction in the QROM with non-standard assumption
- Failure probability of $< 2^{-140}$
- Interesting questions:
 - How much of a problem are a few failures?
 - How much can an attacker exploit Grover to produce failures?
- Three different parameter sets submitted:
 - **Kyber512**: 102 bits of post-quantum security
 - **Kyber768**: 161 bits of post-quantum security
 - **Kyber1024**: 218 bits of post-quantum security



- Tight reduction from MLWE in the ROM (if we didn't compress pk)
- Non-tight reduction in the QROM
- Tight reduction in the QROM with non-standard assumption
- Failure probability of $< 2^{-140}$
- Interesting questions:
 - How much of a problem are a few failures?
 - How much can an attacker exploit Grover to produce failures?
- Three different parameter sets submitted:
 - **Kyber512**: 102 bits of post-quantum security
 - **Kyber768**: 161 bits of post-quantum security
 - **Kyber1024**: 218 bits of post-quantum security
- Security estimates are based on “core-SVP hardness” (see NewHope paper)



Kyber512

Sizes (in bytes)		Haswell cycles (ref)		Haswell cycles (AVX2)
sk:	1632	gen:	141872	gen: 55160
pk:	736	enc:	205468	enc: 75680
ct:	800	dec:	246040	dec: 74428

- Cycle counts on one core, without TurboBoost and HyperThreading
- Comparison: X25519 gen: 90668 cycles, enc/dec: 138963

Kyber768

Sizes (in bytes)		Haswell cycles (ref)		Haswell cycles (AVX2)
sk:	2400	gen:	243004	gen: 85472
pk:	1088	enc:	332616	enc: 112660
ct:	1152	dec:	394424	dec: 108904

- Cycle counts on one core, without TurboBoost and HyperThreading
- Comparison: X25519 gen: 90668 cycles, enc/dec: 138963

Kyber1024

Sizes (in bytes)		Haswell cycles (ref)		Haswell cycles (AVX2)
sk:	3168	gen:	368564	gen: 121056
pk:	1440	enc:	481042	enc: 157964
ct:	1504	dec:	558740	dec: 154952

- Cycle counts on one core, without TurboBoost and HyperThreading
- Comparison: X25519 gen: 90668 cycles, enc/dec: 138963

Kyber1024

Sizes (in bytes)		Haswell cycles (ref)		Haswell cycles (AVX2)
sk:	3168	gen:	368564	gen: 121056
pk:	1440	enc:	481042	enc: 157964
ct:	1504	dec:	558740	dec: 154952

- Cycle counts on one core, without TurboBoost and HyperThreading
- Comparison: X25519 gen: 90668 cycles, enc/dec: 138963
- However, only 32 bytes for X25519 pk and ct



Dilithium: The signature scheme

joint work with

Léo Ducas, Eike Kiltz, Tancrede Lepoint,
Vadim Lyubashevsky, Gregor Seiler, Damien Stehlé



- Use “Fiat-Shamir with aborts” (Lyubashevsky 2009)
- Can think of Dilithium as instantiation of Bai-Gailbraith signatures (2013)



- Use “Fiat-Shamir with aborts” (Lyubashevsky 2009)
- Can think of Dilithium as instantiation of Bai-Gailbraith signatures (2013)
- Avoid Gaussian sampling, use uniform noise
- Reason: easy to implement efficiently



- Use “Fiat-Shamir with aborts” (Lyubashevsky 2009)
- Can think of Dilithium as instantiation of Bai-Gailbraith signatures (2013)
- Avoid Gaussian sampling, use uniform noise
- Reason: easy to implement efficiently
- Optimize for (public-key + signature) size



Fiat-Shamir with aborts (“noisy Schnorr”)

Gen()

$$\mathbf{s}_1, \mathbf{s}_2 \leftarrow S_\eta$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$$

$$pk = (\rho, \mathbf{t}), sk = (\rho, \mathbf{s}_1, \mathbf{s}_2)$$

Verif(pk, m, σ)

$$\mathbf{w}'_1 \leftarrow \mathbf{A}\mathbf{z} - \mathbf{c}\mathbf{t}$$

Verify that $c = H(M, \mathbf{w}'_1)$

Sign(sk, m)

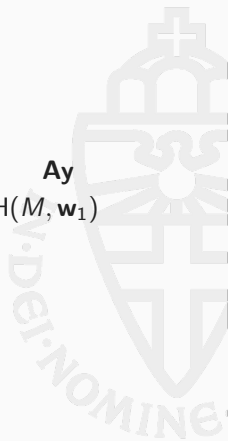
$$\mathbf{y} \leftarrow S_{\gamma_1-1}$$

$$\mathbf{w}_1 \leftarrow$$

$$c \in B_{60} \leftarrow H(M, \mathbf{w}_1)$$

$$\mathbf{z} \leftarrow \mathbf{y} + \mathbf{c}\mathbf{s}_1$$

$$\sigma = (\mathbf{z}, c)$$



Fiat-Shamir with aborts (“noisy Schnorr”)

Gen()

$$\rho \leftarrow \{0, 1\}^{256}$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{s}_1, \mathbf{s}_2 \leftarrow S_\eta$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$$

$$pk = (\rho, \mathbf{t}), sk = (\rho, \mathbf{s}_1, \mathbf{s}_2)$$

Verif(pk, m, σ)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{w}'_1 \leftarrow \mathbf{A}\mathbf{z} - c\mathbf{t}$$

Verify that $c = H(M, \mathbf{w}'_1)$

Sign(sk, m)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

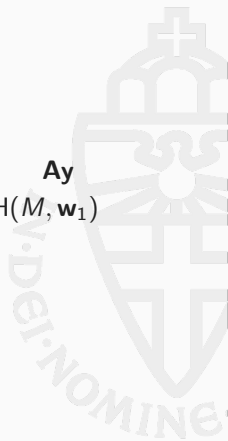
$$\mathbf{y} \leftarrow S_{\gamma_1-1}$$

$$\mathbf{w}_1 \leftarrow$$

$$c \in B_{60} \leftarrow H(M, \mathbf{w}_1)$$

$$\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_1$$

$$\sigma = (\mathbf{z}, c)$$



Fiat-Shamir with aborts (“noisy Schnorr”)

Gen()

$$\rho \leftarrow \{0, 1\}^{256}$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{s}_1, \mathbf{s}_2 \leftarrow S_\eta$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$$

$$pk = (\rho, \mathbf{t}), sk = (\rho, \mathbf{s}_1, \mathbf{s}_2)$$

Verif(pk, m, σ)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{w}'_1 \leftarrow \text{HighBits}(\mathbf{A}\mathbf{z} - \mathbf{c}\mathbf{t}, 2\gamma_2)$$

Verify that $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$

Verify that $c = H(M, \mathbf{w}'_1)$

Sign(sk, m)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{y} \leftarrow S_{\gamma_1 - 1}$$

$$\mathbf{w}_1 \leftarrow \text{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$$

$$c \in B_{60} \leftarrow H(M, \mathbf{w}_1)$$

$$\mathbf{z} \leftarrow \mathbf{y} + \mathbf{c}\mathbf{s}_1$$

$$\sigma = (\mathbf{z}, c)$$

Fiat-Shamir with aborts (“noisy Schnorr”)

Gen()

$$\rho \leftarrow \{0, 1\}^{256}$$

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{s}_1, \mathbf{s}_2 \leftarrow S_\eta$$

$$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$$

$$pk = (\rho, \mathbf{t}), sk = (\rho, \mathbf{s}_1, \mathbf{s}_2)$$

Verif(pk, m, σ)

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{w}'_1 \leftarrow \text{HighBits}(\mathbf{A}\mathbf{z} - \mathbf{c}\mathbf{t}, 2\gamma_2)$$

Verify that $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$

Verify that $c = H(M, \mathbf{w}'_1)$

Sign(sk, m)

Repeat:

$$\mathbf{A} \leftarrow \text{XOF}(\rho)$$

$$\mathbf{y} \leftarrow S_{\gamma_1 - 1}$$

$$\mathbf{w}_1 \leftarrow \text{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$$

$$c \in B_{60} \leftarrow H(M, \mathbf{w}_1)$$

$$\mathbf{z} \leftarrow \mathbf{y} + \mathbf{c}\mathbf{s}_1$$

RejSample(\mathbf{z})

RejSample($\mathbf{A}\mathbf{y} - \mathbf{c}\mathbf{s}_2$)

$$\sigma = (\mathbf{z}, c)$$

- Attacker knows that coefficients of \mathbf{y} are in $\{-\gamma_1 + 1, \dots, \gamma_1 - 1\}$
- What if a coefficient of $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ is close to the border?
- Answer: attacker learns something about \mathbf{s}_1 !



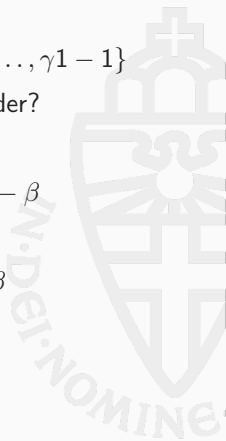
- Attacker knows that coefficients of \mathbf{y} are in $\{-\gamma_1 + 1, \dots, \gamma_1 - 1\}$
- What if a coefficient of $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ is close to the border?
- Answer: attacker learns something about \mathbf{s}_1 !
- Solution: reject if any coefficient of \mathbf{z} is larger than $\gamma_1 - \beta$



- Attacker knows that coefficients of \mathbf{y} are in $\{-\gamma_1 + 1, \dots, \gamma_1 - 1\}$
- What if a coefficient of $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ is close to the border?
- Answer: attacker learns something about \mathbf{s}_1 !
- Solution: reject if any coefficient of \mathbf{z} is larger than $\gamma_1 - \beta$
- Obvious safe value for β : $60 \cdot \eta$



- Attacker knows that coefficients of \mathbf{y} are in $\{-\gamma_1 + 1, \dots, \gamma_1 - 1\}$
- What if a coefficient of $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ is close to the border?
- Answer: attacker learns something about \mathbf{s}_1 !
- Solution: reject if any coefficient of \mathbf{z} is larger than $\gamma_1 - \beta$
- Obvious safe value for β : $60 \cdot \eta$
- Similar: restart if $\|\text{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$
- This second rejection is also required for correctness



- During verification, compute $\mathbf{w}'_1 \leftarrow \text{HighBits}(\mathbf{Az} - \mathbf{ct}, 2\gamma_2)$
- This does not really need the low bits of \mathbf{t}
- Only requires the carries of \mathbf{ct} into the high bits



- During verification, compute $\mathbf{w}'_1 \leftarrow \text{HighBits}(\mathbf{Az} - \mathbf{ct}, 2\gamma_2)$
- This does not really need the low bits of \mathbf{t}
- Only requires the carries of \mathbf{ct} into the high bits
- Idea: compress public key by only including the high bits
- Signature needs to include carries of \mathbf{ct} into high bits



- During verification, compute $\mathbf{w}'_1 \leftarrow \text{HighBits}(\mathbf{Az} - \mathbf{ct}, 2\gamma_2)$
- This does not really need the low bits of \mathbf{t}
- Only requires the carries of \mathbf{ct} into the high bits
- Idea: compress public key by only including the high bits
- Signature needs to include carries of \mathbf{ct} into high bits
- Interesting when public-key size matters:
 - factor-2.5 size reduction of the public key
 - increase signature size by $\approx 4\%$
- Public-key size matters, for example, in certificates



- Non-tight ROM reduction from MLWE and MSIS
- Tight QROM reduction from MLWE, MSIS, and SelfTargetMSIS



- Non-tight ROM reduction from MLWE and MSIS
- Tight QROM reduction from MLWE, MSIS, and SelfTargetMSIS
- Choose β slightly smaller than 60η
 - Requires careful analysis, no concern in practice
 - Drastically reduces number of repetitions



- Non-tight ROM reduction from MLWE and MSIS
- Tight QROM reduction from MLWE, MSIS, and SelfTargetMSIS
- Choose β slightly smaller than 60η
 - Requires careful analysis, no concern in practice
 - Drastically reduces number of repetitions
- Four parameter sets:
 - Dilithium-weak: 53 bits of post-quantum security
 - Dilithium-medium: 91 bits of post-quantum security
 - Dilithium-recommended: 125 bits of post-quantum security
 - Dilithium-very-high: 158 bits of post-quantum security



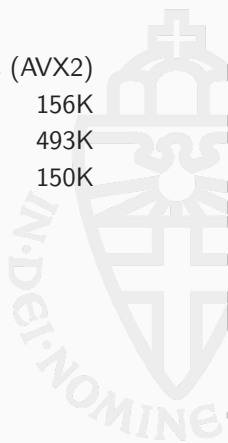
- Non-tight ROM reduction from MLWE and MSIS
- Tight QROM reduction from MLWE, MSIS, and SelfTargetMSIS
- Choose β slightly smaller than 60η
 - Requires careful analysis, no concern in practice
 - Drastically reduces number of repetitions
- Four parameter sets:
 - Dilithium-weak: 53 bits of post-quantum security
 - Dilithium-medium: 91 bits of post-quantum security
 - Dilithium-recommended: 125 bits of post-quantum security
 - Dilithium-very-high: 158 bits of post-quantum security
- Again, use core-SVP hardness of MLWE/MSIS



Dilithium-1024x768 (medium)

Sizes (in bytes)	Haswell cycles (ref)	Haswell cycles (AVX2)
sk: 2800	gen: 269K	gen: 156K
pk: 1184	sign: 1285K	sign: 493K
sig: 2044	verify: 296K	verify: 150K

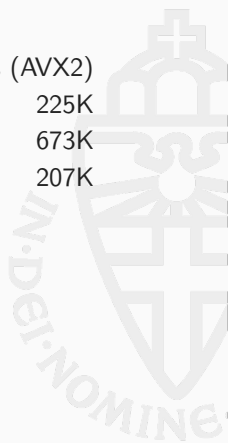
- Comparison with Ed25519:
 - Cycles for signing: 61212
 - Cycles for verification: 182812
 - Signature bytes: 64



Dilithium-1280x1024

Sizes (in bytes)	Haswell cycles (ref)	Haswell cycles (AVX2)
sk: 3504	gen: 382K	gen: 225K
pk: 1472	sign: 1817K	sign: 673K
sig: 2701	verify: 395K	verify: 207K

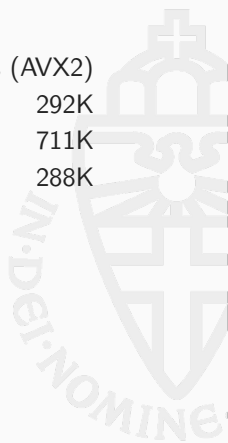
- Comparison with Ed25519:
 - Cycles for signing: 61212
 - Cycles for verification: 182812
 - Signature bytes: 64



Dilithium-1536x1280

Sizes (in bytes)	Haswell cycles (ref)	Haswell cycles (AVX2)
sk: 3856	gen: 512K	gen: 292K
pk: 1760	sign: 1677K	sign: 711K
sig: 3366	verify: 548K	verify: 288K

- Comparison with Ed25519:
 - Cycles for signing: 61212
 - Cycles for verification: 182812
 - Signature bytes: 64



Encryption and KEMs

- If you don't care about public-key size: use McEliece



Encryption and KEMs

- If you don't care about public-key size: use McEliece
- If you do care about public-key size: use Kyber768 or NewHope



Encryption and KEMs

- If you don't care about public-key size: use McEliece
- If you do care about public-key size: use Kyber768 or NewHope
- Combine with pre-quantum crypto, e.g., X25519



Encryption and KEMs

- If you don't care about public-key size: use McEliece
- If you do care about public-key size: use Kyber768 or NewHope
- Combine with pre-quantum crypto, e.g., X25519

Signatures

- If you can, use forward-secure stateful hash-based signatures (XMSS-SHA3)

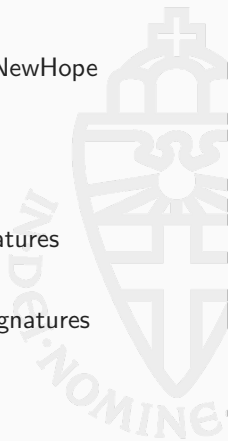


Encryption and KEMs

- If you don't care about public-key size: use McEliece
- If you do care about public-key size: use Kyber768 or NewHope
- Combine with pre-quantum crypto, e.g., X25519

Signatures

- If you can, use forward-secure stateful hash-based signatures (XMSS-SHA3)
- Elseif you can, use (large, slow) stateless hash-based signatures (SPHINCS+-SHA3)

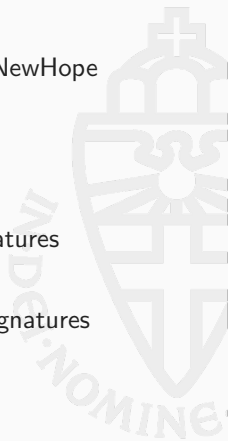


Encryption and KEMs

- If you don't care about public-key size: use McEliece
- If you do care about public-key size: use Kyber768 or NewHope
- Combine with pre-quantum crypto, e.g., X25519

Signatures

- If you can, use forward-secure stateful hash-based signatures (XMSS-SHA3)
- Elseif you can, use (large, slow) stateless hash-based signatures (SPHINCS+-SHA3)
- Else use Dilithium-recommended plus Ed25519



- Personal website: <https://cryptojedi.org>
- CRYSTALS website: <https://pq-crystals.org>

