MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

An Introduction to Lattice-based KEMs

Peter Schwabe

December 7, 2021

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**

4 31 27

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

The NIST competition: round 2

- Announcement planned at Real-World Crypto 2019

The NIST competition: round 2

- Announcement planned at Real-World Crypto 2019
- Due to US government lockdown slightly later

The NIST competition: round 2

- Announcement planned at Real-World Crypto 2019
- Due to US government lockdown slightly later

Encryption / Key agreement

- 9 lattice-based
- 7 code-based
- 1 isogeny-based

The NIST competition: round 2

- Announcement planned at Real-World Crypto 2019
- Due to US government lockdown slightly later

Encryption / Key agreement

- 9 lattice-based
- 7 code-based
- 1 isogeny-based

Signature schemes

- 3 lattice-based
- 2 symmetric-crypto based
- 4 MQ-based

The NIST competition: round 3

- Announcement planned for June 2020

The NIST competition: round 3

- Announcement planned for June 2020
- Due to pandemic (?) slightly later

The NIST competition: round 3

- Announcement planned for June 2020
- Due to pandemic (?) slightly later

Finalists

- 4 key-agreement schemes
  - 3 lattice-based
  - 1 code-based
- 3 signature schemes
  - 2 lattice-based
  - 1 MQ-based

The NIST competition: round 3

- Announcement planned for June 2020
- Due to pandemic (?) slightly later

Finalists

- 4 key-agreement schemes
  - 3 lattice-based
  - 1 code-based
- 3 signature schemes
  - 2 lattice-based
  - 1 MQ-based

Alternate schemes

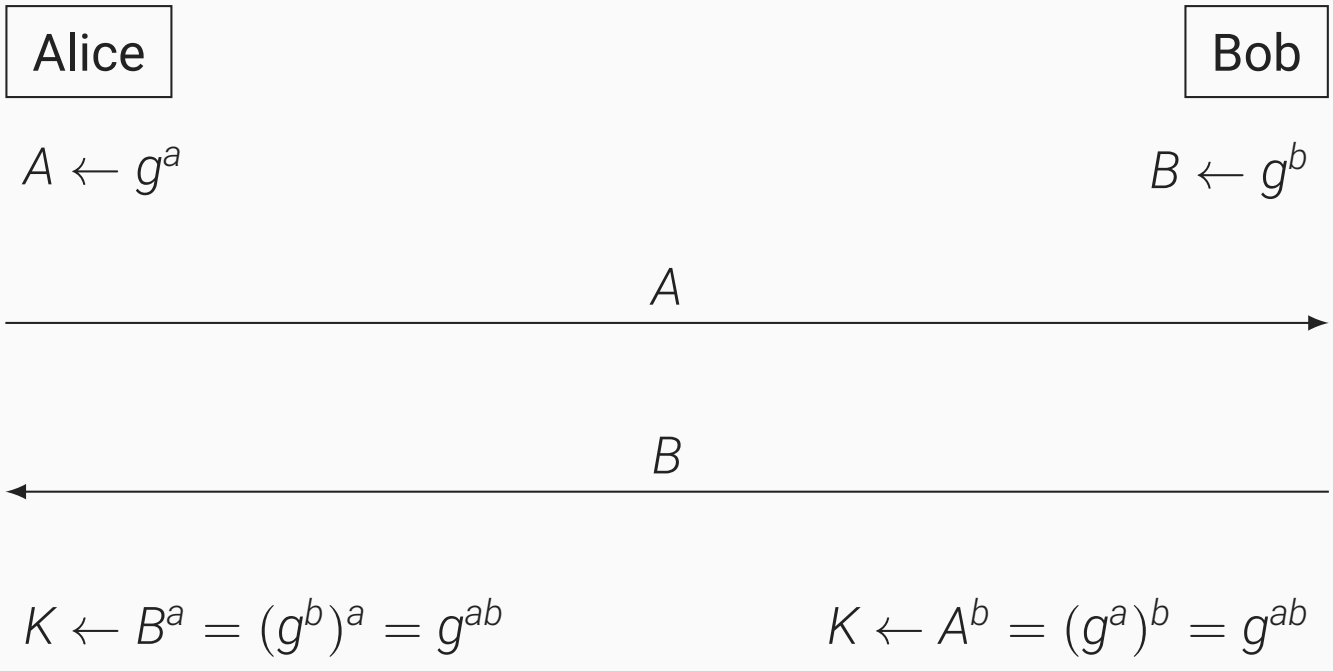
- 5 key-agreement schemes
  - 2 lattice-based
  - 2 code-based
  - 1 isogeny-based
- 3 signature schemes
  - 2 symmetric-crypto based
  - 1 MQ-based

What NIST means by “Key exchange”

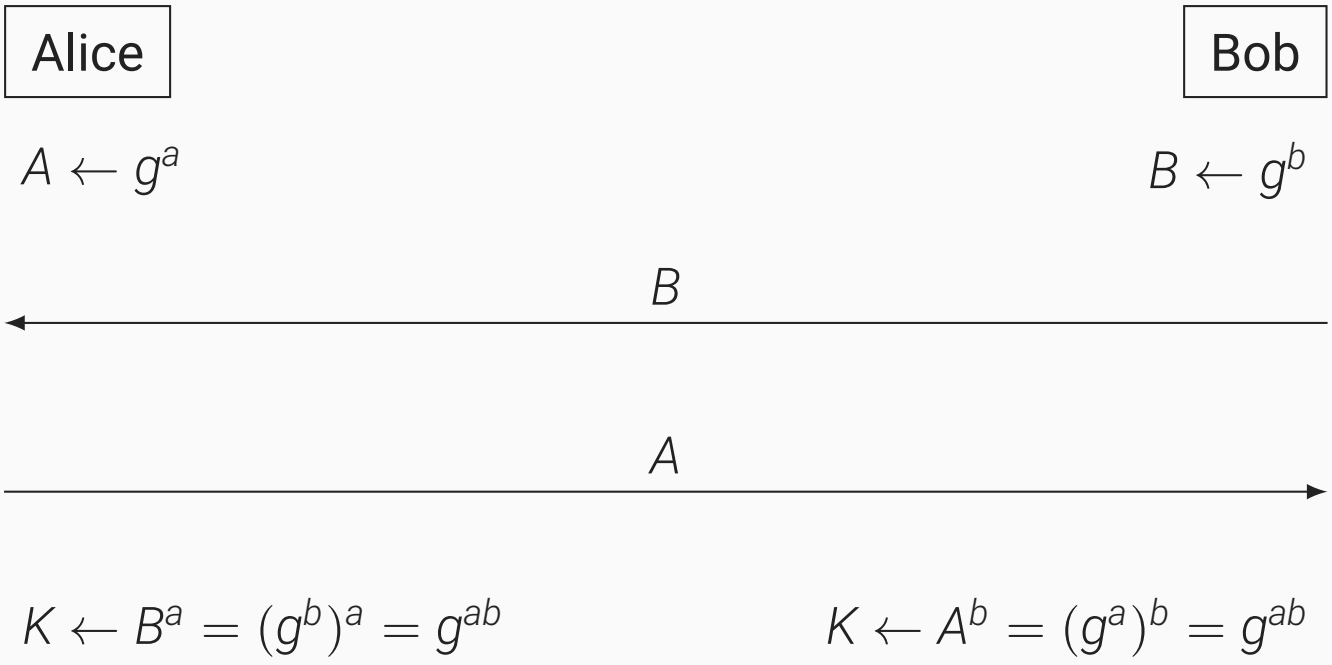
Key encapsulation mechanisms (KEMs)

- $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}()$
- $(c, k) \leftarrow \text{Encaps}(\text{pk})$
- $k \leftarrow \text{Decaps}(c, \text{sk})$

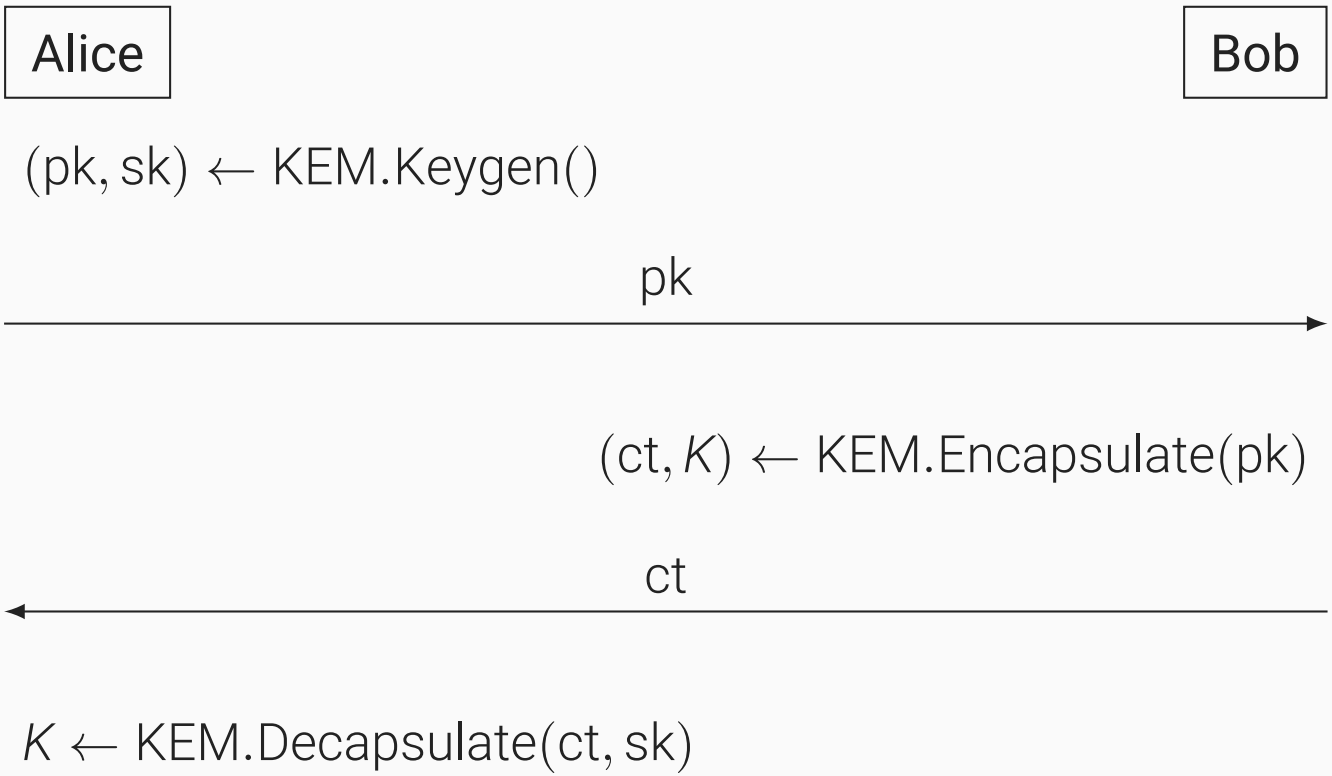
A reminder of Diffie-Hellman



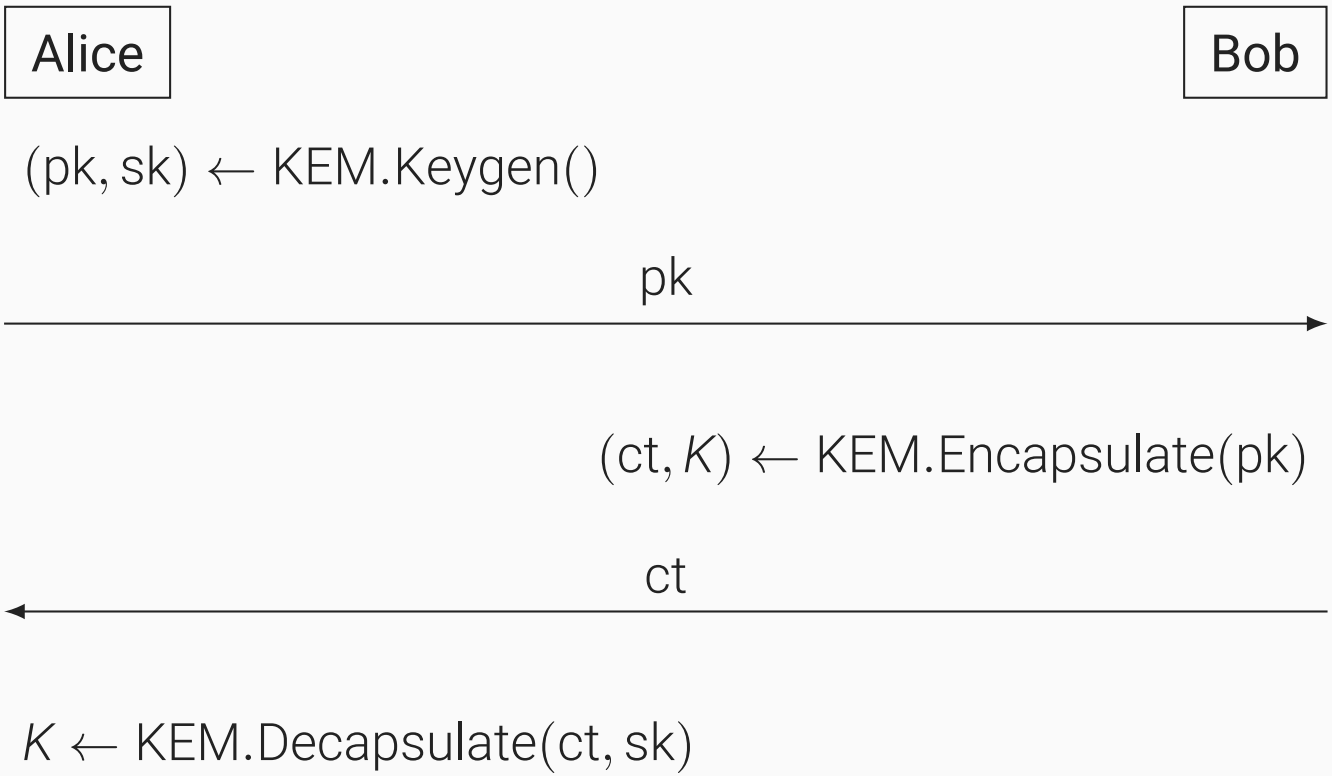
A reminder of Diffie-Hellman



KEMs: as close as you'll get to DH



KEMs: as close as you'll get to DH*

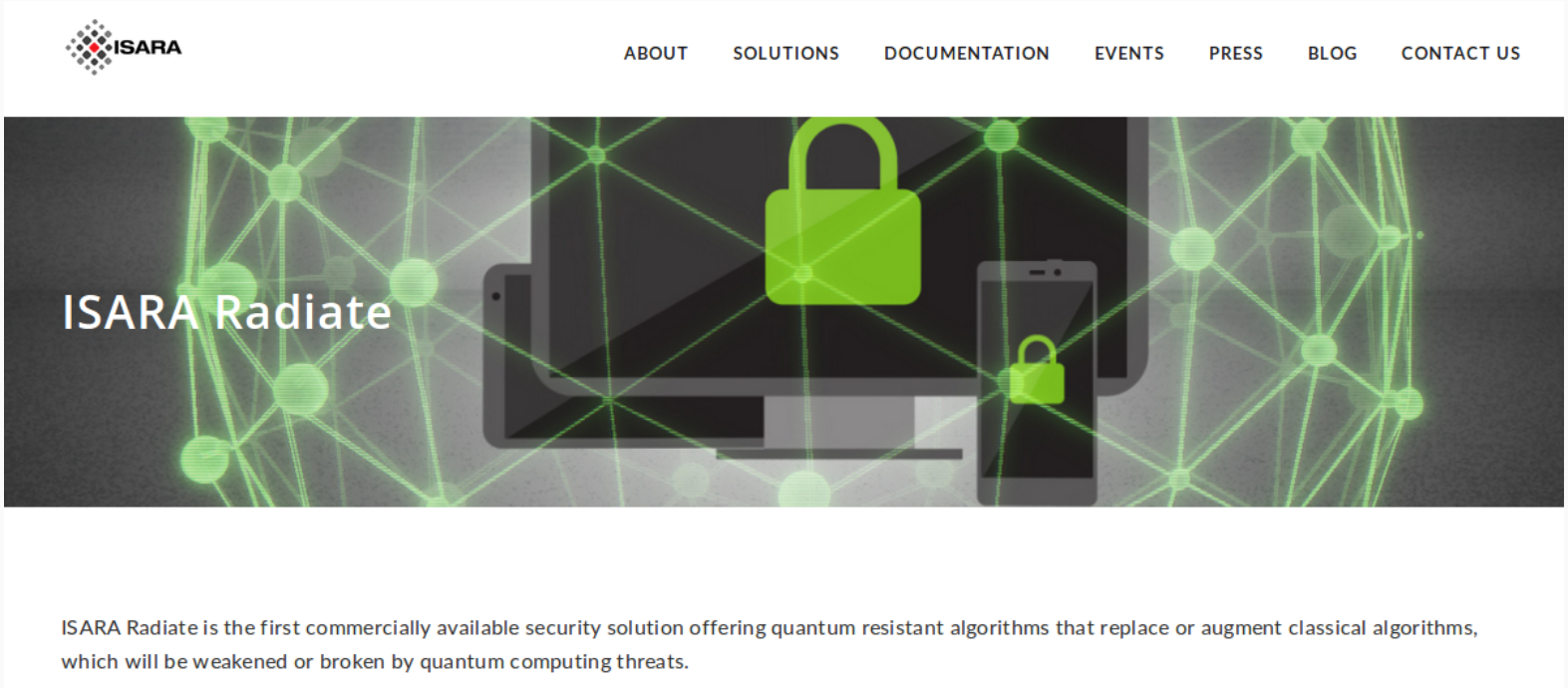


*Except with CSIDH (Castryck, Lange, Martindale, Renes, Panny, 2018)

Lattice-based KEMs

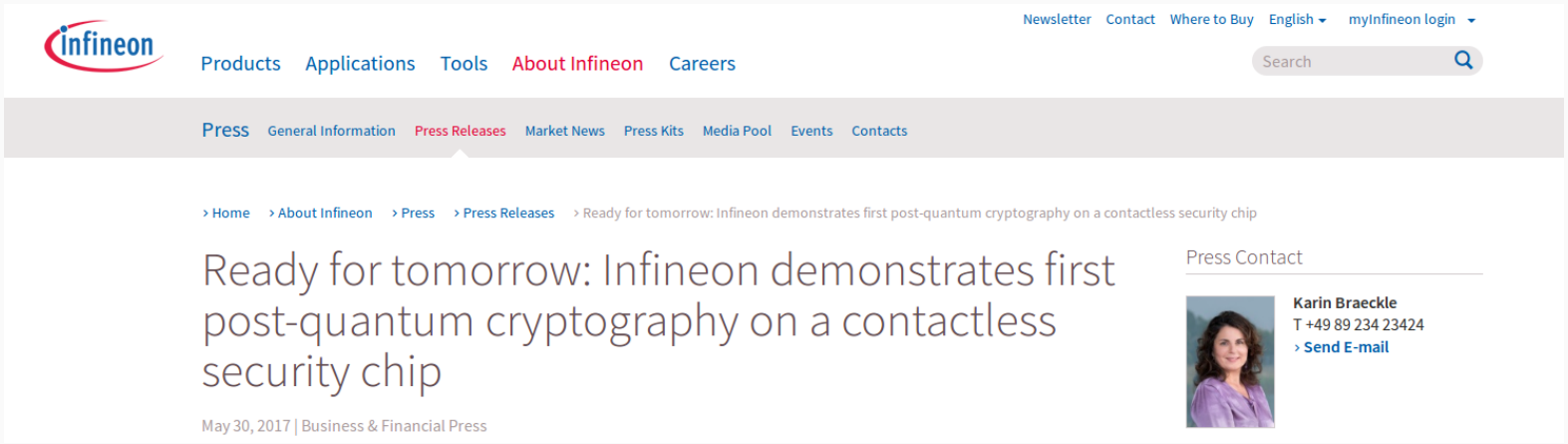
"We're indebted to Erdem Alkim, Léo Ducas, Thomas Pöppelmann and Peter Schwabe, the researchers who developed "New Hope", the post-quantum algorithm that we selected for this experiment."

https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html



"Key Agreement using the 'NewHope' lattice-based algorithm detailed in the New Hope paper, and LUKE (Lattice-based Unique Key Exchange), an ISARA speed-optimized version of the NewHope algorithm."

https://www.isara.com/isara-radiate/



"The deployed algorithm is a variant of "New Hope", a quantum-resistant cryptosystem"

https://www.infineon.com/cms/en/about-infineon/press/press-releases/2017/INFCCS201705-056.html

Learning with errors (LWE)

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

Learning with errors (LWE)

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

Learning with rounding (LWR)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Given samples $\lceil \mathbf{As} \rfloor_p$, with $p < q$

Learning with rounding (LWR)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Given samples $\lceil \mathbf{As} \rfloor_p$, with $p < q$
- Search version: find $\mathbf{s}$
- Decision version: distinguish from uniform random

Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM

Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
- Idea to solve this: allow structured matrix $\mathbf{A}$, e.g.,

Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
- Idea to solve this: allow structured matrix $\mathbf{A}$, e.g.,
  - NewHope: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$; n a power of 2, q prime

Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
- Idea to solve this: allow structured matrix $\mathbf{A}$, e.g.,
  - NewHope: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$; n a power of 2, q prime
  - NTRU: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1)$; n prime, q a power of 2

Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
- Idea to solve this: allow structured matrix $\mathbf{A}$, e.g.,
  - NewHope: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$; n a power of 2, q prime
  - NTRU: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1)$; n prime, q a power of 2
  - NTRU Prime: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - X - 1)$; q prime, n prime

Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
- Idea to solve this: allow structured matrix $\mathbf{A}$, e.g.,
  - NewHope: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$; n a power of 2, q prime
  - NTRU: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1)$; n prime, q a power of 2
  - NTRU Prime: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - X - 1)$; q prime, n prime
  - Kyber/Saber: use small-dimension matrices and vectors over $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^{256} + 1)$

Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
- Idea to solve this: allow structured matrix $\mathbf{A}$, e.g.,
  - NewHope: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$; n a power of 2, q prime
  - NTRU: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1)$; n prime, q a power of 2
  - NTRU Prime: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - X - 1)$; q prime, n prime
  - Kyber/Saber: use small-dimension matrices and vectors over $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^{256} + 1)$
- Perform arithmetic on (vectors of) polynomials instead of vectors/matrices over $\mathbb{Z}_q$

How to build a KEM?

Alice (server)		Bob (client)
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{\mathbf{b}}$	$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
	$\xleftarrow{\mathbf{u}}$	

Alice has $\mathbf{v} = \mathbf{us} = \mathbf{ass}' + \mathbf{e}'\mathbf{s}$

Bob has $\mathbf{v}' = \mathbf{bs}' = \mathbf{ass}' + \mathbf{es}'$

- Secret and noise polynomials $\mathbf{s}, \mathbf{s}', \mathbf{e}, \mathbf{e}'$ are small
- $\mathbf{v}$ and $\mathbf{v}'$ are *approximately* the same

How to build a KEM, part 2

Alice		Bob
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b})}$	
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}'$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u})}$	

How to build a KEM, part 2

Alice		Bob
$\mathit{seed} \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{XOF}(\mathit{seed}))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, \mathit{seed})}$	$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{XOF}(\mathit{seed}))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}'$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u} \quad)}$	

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}'$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		$\mu \leftarrow \text{Extract}(\mathbf{k})$
$\mu \leftarrow \text{Extract}(\mathbf{k}')$		

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		$\mu \leftarrow \text{Extract}(\mathbf{k})$
$\mu \leftarrow \text{Extract}(\mathbf{k}')$		

This is LPR encryption, written as KEM (except for generation of $\mathbf{a}$)

Encode and Extract

- Encoding in LPR encryption: map n bits to n coefficients:
  - A zero bit maps to 0
  - A one bit maps to $q/2$
- Idea: Noise affects low bits of coefficients, put data into high bits

Encode and Extract

- Encoding in LPR encryption: map n bits to n coefficients:
  - A zero bit maps to 0
  - A one bit maps to $q/2$
- Idea: Noise affects low bits of coefficients, put data into high bits
- Decode: map coefficient into $[-q/2, q/2]$
  - Closer to 0 (i.e., in $[-q/4, q/4]$): set bit to zero
  - Closer to $\pm q/2$: set bit to one

From passive to CCA security

- The base scheme does not have active security
- Attacker can choose arbitrary noise, learns $\mathbf{s}$ from failures

From passive to CCA security

- The base scheme does not have active security
- Attacker can choose arbitrary noise, learns $\mathbf{s}$ from failures
- Fujisaki-Okamoto transform (sketched):

Alice (Server)		Bob (Client)
Gen(): pk, sk$\leftarrow$KeyGen() seed, $\mathbf{b}\leftarrow$pk	$\overset{\text{seed},\mathbf{b}}{\rightarrow}$	Enc(seed, $\mathbf{b}$): $x\leftarrow\{0,\ldots,255\}^{32}$ k, coins$\leftarrow$SHA3-512(x)
	$\overset{\mathbf{u},v}{\leftarrow}$	$\mathbf{u}, v\leftarrow$Encrypt((seed, $\mathbf{b}$), x, coins)
Dec($\mathbf{s}$, ($\mathbf{u}$, v)): $x'\leftarrow$ Decrypt($\mathbf{s}$, ($\mathbf{u}$, v)) k', $coins'\leftarrow$SHA3-512(x') $\mathbf{u}'$, $v'\leftarrow$Encrypt((seed, $\mathbf{b}$), x', coins') **verify if** $(\mathbf{u}', v') = (\mathbf{u}, v)$		

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- Decrypt:
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e}$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- Decrypt:
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m})$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- Decrypt:
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) = \mathbf{f}(\mathbf{r} \cdot (p\mathbf{f}_q\mathbf{g}) + \mathbf{m})$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- Decrypt:
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) = \mathbf{f}(\mathbf{r} \cdot (p\mathbf{f}_q\mathbf{g}) + \mathbf{m}) = p\mathbf{r}\mathbf{g} + \mathbf{f} \cdot \mathbf{m}$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- Decrypt:
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) = \mathbf{f}(\mathbf{r} \cdot (p\mathbf{f}_q\mathbf{g}) + \mathbf{m}) = p\mathbf{r}\mathbf{g} + \mathbf{f} \cdot \mathbf{m}$
  - Compute $\mathbf{m} = \mathbf{v} \cdot \mathbf{f}_p \mod p$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- Decrypt:
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) = \mathbf{f}(\mathbf{r} \cdot (p\mathbf{f}_q\mathbf{g}) + \mathbf{m}) = p\mathbf{r}\mathbf{g} + \mathbf{f} \cdot \mathbf{m}$
  - Compute $\mathbf{m} = \mathbf{v} \cdot \mathbf{f}_p \mod p$
- Advantages/Disadvantages compared to LPR:
  - Asymptotically weaker than Ring-LWE approach
  - Slower keygen, but faster encryption/decryption

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- First option: $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- Second option: $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- First option: $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- Second option: $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- Third option: $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1) = \Phi_{2n}$, $n = 2^m$ (NewHope, Kyber, LAC)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1) = \Phi_{2n}$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1) = \Phi_{2n}$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1) = \Phi_{2n}$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1) = \Phi_{2n}$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure
- NTRU Prime advertises “less structure” in their $\mathcal{R}_q$

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1) = \Phi_{2n}$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure
- NTRU Prime advertises “less structure” in their $\mathcal{R}_q$
- NewHope and Kyber have fastest (NTT-based) arithmetic

Design space 2: module vs. ring?

- "Traditionally", work directly with elements of $\mathcal{R}_q$ ("Ring-LWE")
- Alternative: Module-LWE (MLWE):
  - Choose smaller n, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$

Design space 2: module vs. ring?

- "Traditionally", work directly with elements of $\mathcal{R}_q$ ("Ring-LWE")
- Alternative: Module-LWE (MLWE):
  - Choose smaller n, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$
- MLWE encrypts shorter messages than Ring-LWE

Design space 2: module vs. ring?

- "Traditionally", work directly with elements of $\mathcal{R}_q$ ("Ring-LWE")
- Alternative: Module-LWE (MLWE):
  - Choose smaller n, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$
- MLWE encrypts shorter messages than Ring-LWE
- MLWE eliminates some of the structure of Ring-LWE

Design space 2: module vs. ring?

- "Traditionally", work directly with elements of $\mathcal{R}_q$ ("Ring-LWE")
- Alternative: Module-LWE (MLWE):
  - Choose smaller n, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$
- MLWE encrypts shorter messages than Ring-LWE
- MLWE eliminates some of the structure of Ring-LWE
- MLWE can very easily scale security (change dimension of matrix):
  - Optimize arithmetic in $\mathcal{R}_q$ once
  - Use same optimized $\mathcal{R}_q$ arithmetic for all security levels

Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
  - more security from the underlying hard problem
  - higher failure probability of decryption

Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
  - more security from the underlying hard problem
  - higher failure probability of decryption
- Three main choices to make:
  - **Narrow or wide noise**
    - Narrow noise (e.g., in $\{-1, 0, 1\}$) not conservative
    - Wide noise requires larger q (or more failures)
    - Larger q means larger public key and ciphertext

Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
  - more security from the underlying hard problem
  - higher failure probability of decryption
- Three main choices to make:
  - **Narrow or wide noise**
    - Narrow noise (e.g., in $\{-1, 0, 1\}$) not conservative
    - Wide noise requires larger q (or more failures)
    - Larger q means larger public key and ciphertext
  - **LWE or LWR**
    - LWE considered more conservative (independent noise)
    - LWR easier to implement (no noise sampling)
    - LWR allows more compact public key and ciphertext

Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
  - more security from the underlying hard problem
  - higher failure probability of decryption
- Three main choices to make:
  - **Narrow or wide noise**
    - Narrow noise (e.g., in $\{-1, 0, 1\}$) not conservative
    - Wide noise requires larger q (or more failures)
    - Larger q means larger public key and ciphertext
  - **LWE or LWR**
    - LWE considered more conservative (independent noise)
    - LWR easier to implement (no noise sampling)
    - LWR allows more compact public key and ciphertext
  - **Fixed-weight noise or not?**
    - Fixed-weight noise needs random permutation (sorting)
    - Naive implementations leak secrets through timing
    - Advantage of fixed-weight: easier to bound (or eliminate) decryption failures

Design space 4: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger q)

Design space 4: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger q)
- For passive-security-only can go the other way:
  - Allow failure probability of, e.g., 2^{-30}
  - Reduce size of public key and ciphertext

Design space 4: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger q)
- For passive-security-only can go the other way:
  - Allow failure probability of, e.g., 2^{-30}
  - Reduce size of public key and ciphertext
- Active (CCA) security needs negligible failure probability

Design space 5: public parameters?

- "Traditional" approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  "Let $\mathbf{a}$ *be a uniformly random. . ."*

Design space 5: public parameters?

- "Traditional" approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  "Let $\mathbf{a}$ be a uniformly random. . . "

- Before NewHope: *real-world* approach: generate fixed $\mathbf{a}$ once

Design space 5: public parameters?

- "Traditional" approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  "Let $\mathbf{a}$ be a uniformly random. . . "

- Before NewHope: *real-world* approach: generate fixed $\mathbf{a}$ once
- What if $\mathbf{a}$ is backdoored?
- Parameter-generating authority can break key exchange
- "Solution": Nothing-up-my-sleeves (involves endless discussion!)

Design space 5: public parameters?

- "Traditional" approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  "Let $\mathbf{a}$ be a uniformly random. . . "

- Before NewHope: *real-world* approach: generate fixed $\mathbf{a}$ once
- What if $\mathbf{a}$ is backdoored?
- Parameter-generating authority can break key exchange
- "Solution": Nothing-up-my-sleeves (involves endless discussion!)
- Even without backdoor:
  - Perform massive precomputation based on $\mathbf{a}$
  - Use precomputation to break *all* key exchanges
  - Infeasible today, but who knows. . .
  - Attack in the spirit of Logjam

Design space 5: public parameters?

- "Traditional" approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  "Let $\mathbf{a}$ be a uniformly random. . . "

- Before NewHope: *real-world* approach: generate fixed $\mathbf{a}$ once
- What if $\mathbf{a}$ is backdoored?
- Parameter-generating authority can break key exchange
- "Solution": Nothing-up-my-sleeves (involves endless discussion!)
- Even without backdoor:
  - Perform massive precomputation based on $\mathbf{a}$
  - Use precomputation to break *all* key exchanges
  - Infeasible today, but who knows. . .
  - Attack in the spirit of Logjam
- Solution in NewHope: Choose a fresh $\mathbf{a}$ every time
- Server can cache $\mathbf{a}$ for some time (e.g., 1h)
- All NIST PQC candidates now use this approach

Design space 6: error-correcting codes?

- Ring-LWE/LWR schemes work with polynomials of > 256 coefficients
- "Encrypt" messages of > 256 bits
- **Need to encrypt** only 256-bit key
- Question: How do we put those additional bits to use?
- Answer: Use error-correcting code (ECC) to reduce failure probability

Design space 6: error-correcting codes?

- Ring-LWE/LWR schemes work with polynomials of > 256 coefficients
- "Encrypt" messages of > 256 bits
- **Need to encrypt** only 256-bit key
- Question: How do we put those additional bits to use?
- Answer: Use error-correcting code (ECC) to reduce failure probability
- NewHope: very simple threshold decoding

Design space 6: error-correcting codes?

- Ring-LWE/LWR schemes work with polynomials of > 256 coefficients
- "Encrypt" messages of > 256 bits
- **Need to encrypt** only 256-bit key
- Question: How do we put those additional bits to use?
- Answer: Use error-correcting code (ECC) to reduce failure probability
- NewHope: very simple threshold decoding
- LAC, Round5: more advanced ECC
  - Correct more errors, obtain smaller public key and ciphertext
  - More complex to implement, in particular without leaking through timing

Design space 7: CCA security?

- Ephemeral key exchange does not need CCA security
- Can offer passively secure version
- Protocols will combine this with signatures for authentication

Design space 7: CCA security?

- Ephemeral key exchange does not need CCA security
- Can offer passively secure version
- Protocols will combine this with signatures for authentication
- Advantages:
  - Higher failure probability $\rightarrow$ more compact
  - Simpler to implement, no CCA transform
  - More flexibility for secret/noise generation

Design space 7: CCA security?

- Ephemeral key exchange does not need CCA security
- Can offer passively secure version
- Protocols will combine this with signatures for authentication
- Advantages:
  - Higher failure probability $\rightarrow$ more compact
  - Simpler to implement, no CCA transform
  - More flexibility for secret/noise generation
- Disadvantages:
  - Less robust (will somebody reuse keys?)
  - More options (CCA vs. CPA): easier to make mistakes

Design space 8: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)

Design space 8: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
  - Hash public-key into shared key: KEM becomes contributory

Design space 8: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
  - Hash public-key into shared key: KEM becomes contributory
  - Hash ciphertext into shared key: more robust (?)

Design space 8: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
  - Hash public-key into shared key: KEM becomes contributory
  - Hash ciphertext into shared key: more robust (?)
- How to handle rejection?
  - Return special symbol (`return -1`): explicit
  - Return H(s, C) for secret s: implicit

Design space 8: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
  - Hash public-key into shared key: KEM becomes contributory
  - Hash ciphertext into shared key: more robust (?)
- How to handle rejection?
  - Return special symbol (`return -1`): explicit
  - Return H(s, C) for secret s: implicit
- As of round 2, no proposal uses explicit rejection
  - Would break some security reduction
  - More robust in practice (return value alwas `0`)

Summary

- Lattice-based KEMs offer best overall performance in the PQ world
- Many tradeoffs between
  - Security (including passive vs. active)
  - Failure rate
  - Size
  - Speed
- More information about NIST PQC:
  - https://csrc.nist.gov/projects/post-quantum-cryptography
  - https://pqc-wiki.fau.edu/