Radboud University

The NIST post-quantum project

Peter Schwabe
peter@cryptojedi.org
https://cryptojedi.org

September 4, 2019

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)

Crypto today

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)

Crypto today

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.

Will there be quantum computers?

"In the past, people have said, maybe it's 50 years away, it's a dream, maybe it'll happen sometime. I used to think it was 50. Now I'm thinking like it's 15 or a little more. It's within reach. It's within our lifetime. It's going to happen."

—Mark Ketchen (IBM), Feb. 2012, about quantum computers

Definition

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

Post-quantum crypto

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 PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012

The NIST PQC "not-a-competition"

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST

The NIST PQC "not-a-competition"

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST
- Widely successful in the past, but also some criticism:
  - Small tweaks are typically allowed, but standardized scheme represents state of the art *at the beginning of the competition*

The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST
- Widely successful in the past, but also some criticism:
  - Small tweaks are typically allowed, but standardized scheme represents state of the art *at the beginning of the competition*
  - AES standardization unaware of cache-timing vulnerabilities

The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST
- Widely successful in the past, but also some criticism:
  - Small tweaks are typically allowed, but standardized scheme represents state of the art *at the beginning of the competition*
  - AES standardization unaware of cache-timing vulnerabilities
  - SHA-3 criterion of 512-bit preimage security unnecessary

The NIST PQC "not-a-competition"

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST
- Widely successful in the past, but also some criticism:
  - Small tweaks are typically allowed, but standardized scheme represents state of the art *at the beginning of the competition*
  - AES standardization unaware of cache-timing vulnerabilities
  - SHA-3 criterion of 512-bit preimage security unnecessary
- PQC project:
  - Announcement: Feb 2016
  - Call for proposals: Dec 2016 (based on community input)
  - Deadline for submissions: Nov 2017

The NIST PQC “not-a-competition” ctd.

Submission categories

- Cryptographic signatures (only stateless)
  - Security for at least 2^{64} signatures per key
- Public-key encryption / key encapsulation
  - Passive or active security (CPA or CCA2)

The NIST PQC "not-a-competition" ctd.

Submission categories

- Cryptographic signatures (only stateless)
  - Security for at least 2^{64} signatures per key
- Public-key encryption / key encapsulation
  - Passive or active security (CPA or CCA2)

Security categories

- Level 1: Equivalent to AES-128 (pre- and post-quantum)
- Level 2: Equivalent to SHA-256 (pre- and post-quantum)
- Level 3: Equivalent to AES-192 (pre- and post-quantum)
- Level 4: Equivalent to SHA-512 (pre- and post-quantum)
- Level 5: Equivalent to AES-256 (pre- and post-quantum)

The NIST competition, initial overview

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.

"Key exchange"

- What is meant is **key encapsulation mechanisms** (KEMs)
  - $(\mathrm{vk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}()$
  - $(c, k) \leftarrow \mathrm{Encaps}(\mathrm{vk})$
  - $k \leftarrow \mathrm{Decaps}(c, \mathrm{sk})$

The NIST competition (ctd.)

"Key exchange"

- What is meant is **key encapsulation mechanisms** (KEMs)
  - $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$
  - $(c, k) \leftarrow \mathsf{Encaps}(\mathsf{vk})$
  - $k \leftarrow \mathsf{Decaps}(c, \mathsf{sk})$

Status of the NIST competition

- In total 69 submissions accepted as "complete and proper"
- Several broken, 5 withdrawn
- Jan 2019: NIST announces 26 round-2 candidates
  - 17 KEMs and PKEs
  - 9 signature schemes

NIST finalists as drop-in replacements?

- Can wait until NIST standardizes some algorithms in ≈ 5 years
- Plug these algorithms into existing protocols and systems
- My impression: that's what many systems designers expect

NIST finalists as drop-in replacements?

- Can wait until NIST standardizes some algorithms in ≈ 5 years
- Plug these algorithms into existing protocols and systems
- My impression: that's what many systems designers expect
- Message of this talk: **this is a terrible idea!**
- Would generate a generation of rather poor protocols
  - mediocre performance (designed pre-quantum, instantiated post-quantum)
  - Suboptimal security properties

NIST finalists as drop-in replacements?

- Can wait until NIST standardizes some algorithms in ≈ 5 years
- Plug these algorithms into existing protocols and systems
- My impression: that's what many systems designers expect
- Message of this talk: **this is a terrible idea!**
- Would generate a generation of rather poor protocols
  - mediocre performance (designed pre-quantum, instantiated post-quantum)
  - Suboptimal security properties
- Bad crypto is very hard to get rid of (think MD5)
- We probably have *one shot* to get this done properly
  - Systems *will* have to transition to PQ crypto
  - Let's work on getting the best out of this transition!
  - Requires interaction between cryptographers and systems designers

The starting point: ECC

- Today: build asymmetric crypto from elliptic-curve arithmetic
- Given P on a curve, $s \in \mathbb{Z}$, compute $Q = sP$
- ECDLP: hard to compute s, given P and Q

The starting point: ECC

- Today: build asymmetric crypto from elliptic-curve arithmetic
- Given P on a curve, $s \in \mathbb{Z}$, compute $Q = sP$
- ECDLP: hard to compute s, given P and Q
- Use for ECDH for key encapsulation and encryption
- Use for ECDSA or Schnorr signatures
- Use same curves, same parameters

The starting point: ECC

- Today: build asymmetric crypto from elliptic-curve arithmetic
- Given P on a curve, $s \in \mathbb{Z}$, compute $Q = sP$
- ECDLP: hard to compute s, given P and Q
- Use for ECDH for key encapsulation and encryption
- Use for ECDSA or Schnorr signatures
- Use same curves, same parameters
- Performance:
  - All operations between 50 000 and 200 000 cycles
  - Keys and ciphertexts: 32 bytes
  - Signatures: 64 bytes

The starting point: ECC

- Today: build asymmetric crypto from elliptic-curve arithmetic
- Given P on a curve, $s \in \mathbb{Z}$, compute $Q = sP$
- ECDLP: hard to compute s, given P and Q
- Use for ECDH for key encapsulation and encryption
- Use for ECDSA or Schnorr signatures
- Use same curves, same parameters
- Performance:
  - All operations between 50 000 and 200 000 cycles
  - Keys and ciphertexts: 32 bytes
  - Signatures: 64 bytes
- Let's look at post-quantum candidates (at NIST security level 3)

PQ-Signatures, part 1: $\mathcal{MQ}$-based

- Find solution to system of m quadratic eqns in n variables over $\mathbb{F}_q$
- Additional assumption: attacker cannot exploit structure
- No reduction from $\mathcal{MQ}$

PQ-Signatures, part 1: $\mathcal{MQ}$-based

- Find solution to system of m quadratic eqns in n variables over $\mathbb{F}_q$
- Additional assumption: attacker cannot exploit structure
- No reduction from $\mathcal{MQ}$
- Example: NIST candidate GeMSS (others: Rainbow, LUOV)
  - Signing: ≈ 2.7 billion cycles
  - Verification: $\approx 580\,000$ cycles
  - Signature: ≈ 50 bytes
  - Public key: $\approx 1.2\,\mathrm{MB}$

PQ-Signatures, part 1: $\mathcal{MQ}$-based

- Find solution to system of m quadratic eqns in n variables over $\mathbb{F}_q$
- Additional assumption: attacker cannot exploit structure
- No reduction from $\mathcal{MQ}$
- Example: NIST candidate GeMSS (others: Rainbow, LUOV)
  - Signing: $\approx$ 2.7 billion cycles
  - Verification: $\approx$ 580 000 cycles
  - Signature: $\approx$ 50 bytes
  - Public key: $\approx$ 1.2 MB
- Can also construct signatures *with* reduction from $\mathcal{MQ}$
- Example: NIST candidate MQDSS
  - Signing $\approx$ 15 Mio cycles
  - Verification $\approx$ 10 Mio cycles
  - Signature: $\approx$ 60 KB
  - Public key: 64 bytes

PQ-Signatures, part 2: lattice-based

- Based on, e.g., LWE (see later) and SIS

PQ-Signatures, part 2: lattice-based

- Based on, e.g., LWE (see later) and SIS
- All NIST candidates use structured lattices (again, see later)
- Example: Dilithium (others: qTESLA, FALCON)
  - Signing: $\approx 500\,000$ cycles
  - Verification: $\approx 170\,000$ cycles
  - Public key: $\approx 1.5\,\text{KB}$
  - Signature: $\approx 2.7\,\text{KB}$

PQ-Signatures, part 3: symmetric-crypto-based

- NIST round-2 candidates: SPHINCS$^+$ and Picnic

PQ-Signatures, part 3: symmetric-crypto-based

- NIST round-2 candidates: SPHINCS$^+$ and Picnic
- Two hash-based signatures in NIST PQC round 2:
  - SPHINCS$^+$: state-of-the art conservative hash-based
  - Picnic: Fiat-Shamir on top of symmetric ID scheme

PQ-Signatures, part 3: symmetric-crypto-based

- NIST round-2 candidates: SPHINCS$^+$ and Picnic
- Two hash-based signatures in NIST PQC round 2:
  - SPHINCS$^+$: state-of-the art conservative hash-based
  - Picnic: Fiat-Shamir on top of symmetric ID scheme
- Hash-based sigs: many tradeoffs possible between
  - Speed (signing is generally slow)
  - Security (trivially via hash sizes)
  - Size (roughly 10-50 KB)
  - Maxium number of signatures per key

PQ-Signatures, part 3: symmetric-crypto-based

- NIST round-2 candidates: SPHINCS$^+$ and Picnic
- Two hash-based signatures in NIST PQC round 2:
  - SPHINCS$^+$: state-of-the art conservative hash-based
  - Picnic: Fiat-Shamir on top of symmetric ID scheme
- Hash-based sigs: many tradeoffs possible between
  - Speed (signing is generally slow)
  - Security (trivially via hash sizes)
  - Size (roughly 10-50 KB)
  - Maxium number of signatures per key
- Example: SPHINCS$^+$-SHA256-192f-robust
  - Signing: $\approx$ 66 Mio cycles
  - Verification: $\approx$ 9.6 Mio cycles
  - Signature: $\approx$ 35.5 KB
  - Public key: 48 bytes
  - Up to 2^{64} signatures

PQ-KEMs, part 1: code-based

- Idea: Take error-correcting code for up to t errors
- Keep *decoding* algorithm secret
- Encryption: map message to code word, add t errors
- Most prominent example: McEliece (1978), uses binary Goppa codes

PQ-KEMs, part 1: code-based

- Idea: Take error-correcting code for up to t errors
- Keep *decoding* algorithm secret
- Encryption: map message to code word, add t errors
- Most prominent example: McEliece (1978), uses binary Goppa codes
- "Classic McEliece" KEM NIST submission (other: NTS-KEM)
  - Encapsulation: $\approx 90\,000$ cycles
  - Decapsulation: $\approx 270\,000$ cycles
  - Key generation: ≈ 300 Mio cycles
  - Cipher text: 188 bytes
  - Public key: ≈ 0.5 MB

PQ-KEMs, part 1: code-based

- Idea: Take error-correcting code for up to t errors
- Keep *decoding* algorithm secret
- Encryption: map message to code word, add t errors
- Most prominent example: McEliece (1978), uses binary Goppa codes
- "Classic McEliece" KEM NIST submission (other: NTS-KEM)
  - Encapsulation: $\approx 90\,000$ cycles
  - Decapsulation: $\approx 270\,000$ cycles
  - Key generation: ≈ 300 Mio cycles
  - Cipher text: 188 bytes
  - Public key: ≈ 0.5 MB
- Probably good choice for, e.g., GPG, but not for low-latency applications
- Possible solution: use structured codes
  (**NIST candidates**: BIKE, LEDAcrypt, HQC, ROLLO, RQC)
- Less studied, less conservative, often problems with CCA security

PQ-KEMs, part 3: SIKE

- Started as "supersingular-isogeny Diffie-Hellman" (SIDH), Jao, De Feo, 2011
- Given two elliptic curves E, E' from the same isogeny class
- Find path of small isogenies from E to E'
- Security related to claw finding, but no reduction from claw finding

PQ-KEMs, part 3: SIKE

- Started as "supersingular-isogeny Diffie-Hellman" (SIDH), Jao, De Feo, 2011
- Given two elliptic curves E, E' from the same isogeny class
- Find path of small isogenies from E to E'
- Security related to claw finding, but no reduction from claw finding
- Rather young construction, more study needed
- Active attacks in 2016 by Galbraith, Petit, Shani, and Ti
- Secure SIDH (or SIKE) is **not** "analogous to the Diffie-Hellman key exchange"

PQ-KEMs, part 3: SIKE

- Started as "supersingular-isogeny Diffie-Hellman" (SIDH), Jao, De Feo, 2011
- Given two elliptic curves E, E' from the same isogeny class
- Find path of small isogenies from E to E'
- Security related to claw finding, but no reduction from claw finding
- Rather young construction, more study needed
- Active attacks in 2016 by Galbraith, Petit, Shani, and Ti
- Secure SIDH (or SIKE) is **not** "analogous to the Diffie-Hellman key exchange"
- SIKE performance:
  - Keygen: ≈ 2.6 Mio cycles
  - Encaps: ≈ 3.8 Mio cycles
  - Decaps: ≈ 4.5 Mio cycles
  - Public key/ciphertext: < 500 bytes each

PQ-KEMs, part 3: SIKE

- Started as "supersingular-isogeny Diffie-Hellman" (SIDH), Jao, De Feo, 2011
- Given two elliptic curves E, E' from the same isogeny class
- Find path of small isogenies from E to E'
- Security related to claw finding, but no reduction from claw finding
- Rather young construction, more study needed
- Active attacks in 2016 by Galbraith, Petit, Shani, and Ti
- Secure SIDH (or SIKE) is **not** "analogous to the Diffie-Hellman key exchange"
- SIKE performance:
  - Keygen: ≈ 2.6 Mio cycles
  - Encaps: ≈ 3.8 Mio cycles
  - Decaps: ≈ 4.5 Mio cycles
  - Public key/ciphertext: < 500 bytes each
  - Even more compact (and slower) with compression

Lattice-based KEMs

- 9 out of 19 NIST round-2 KEMs are (sort of) lattice based:
    - CRYSTALS-Kyber (short: Kyber)
    - FrodoKEM
    - LAC
    - NewHope
    - NTRU
    - NTRU Prime
    - Round5
    - Saber
    - Threebears
- I'm involved in CRYSTALS-Kyber, NewHope, and NTRU
- Two main reasons for the large number:
    - Large design space with many tradeoffs
    - Popularity before the NIST project (in particular through NewHope)

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Experimenting with Post-Quantum Cryptography

July 7, 2016

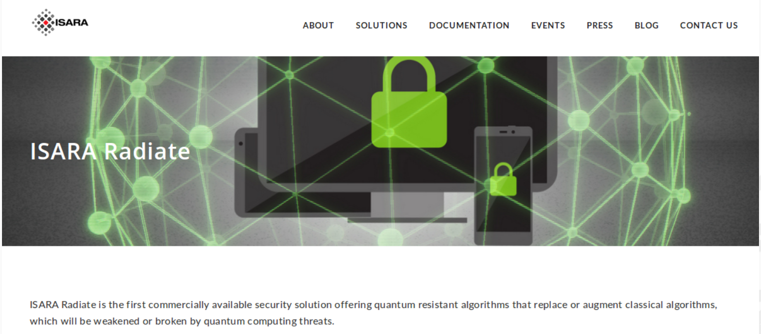
Posted by Matt Braithwaite, Software Engineer

Search blog ...

Archive

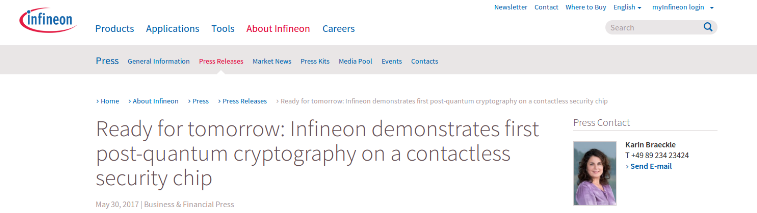
“We’re indebted to Erdem Alkim, Léo Ducas, Thomas Pöppelmann and Peter Schwabe, the researchers who developed “New Hope”, the post-quantum algorithm that we selected for this experiment.”

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

- Problem with LWE-based cryptosystems: public-key size
- Idea to solve this: allow structured matrix $\mathbf{A}$, e.g.,

Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Idea to solve this: allow structured matrix **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
- 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
- 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
- Idea to solve this: allow structured matrix **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
- 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)
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{\mathbf{b}}$	$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
	$\xleftarrow{\mathbf{u}}$	

$$\begin{array}{lllll} \text{Alice has} & \mathbf{v} & = \mathbf{u}\mathbf{s} & = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}'\mathbf{s} \\ \text{Bob has} & \mathbf{v}' & = \mathbf{b}\mathbf{s}' & = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}\mathbf{s}' \end{array}$$

- 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
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}\quad)}$	
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}'$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}\quad)}$	

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathrm{Parse}(\mathrm{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \mathrm{Parse}(\mathrm{SHAKE\text{-}128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}'$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}\quad)}$	

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}'$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \mathsf{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathrm{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \mathrm{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \mathrm{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		$\mu \leftarrow \text{Extract}(\mathbf{k})$
$\mu \leftarrow \text{Extract}(\mathbf{k}')$		

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		$\mu \leftarrow \text{Extract}(\mathbf{k})$
$\mu \leftarrow \text{Extract}(\mathbf{k}')$		

This is LPR encryption, written as KEX (except for generation of **a**)

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)
$\underline{\text{Gen}()}$:		$\underline{\text{Enc}(\text{seed}, \mathbf{b})}$:
$\text{pk}, \text{sk} \leftarrow \text{KeyGen}()$		$x \leftarrow \{0, \ldots, 255\}^{32}$
$\text{seed}, \mathbf{b} \leftarrow \text{pk}$	$\xrightarrow{\text{seed}, \mathbf{b}}$	$x \leftarrow \text{SHA3-256}(x)$
		$k, \text{coins} \leftarrow \text{SHA3-512}(x)$
	$\xleftarrow{\mathbf{u}, v}$	$\mathbf{u}, v \leftarrow \text{Encrypt}((\text{seed}, \mathbf{b}), x, \text{coins})$
$\underline{\text{Dec}(\mathbf{s}, (\mathbf{u}, v))}$:		
$x' \leftarrow \text{Decrypt}(\mathbf{s}, (\mathbf{u}, v))$		
$k', coins' \leftarrow \text{SHA3-512}(x')$		
$\mathbf{u}', v' \leftarrow \text{Encrypt}((\text{seed}, \mathbf{b}), x', \text{coins}')$		
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 1: 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 1: 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 1: 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 1: 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 prob.

Design space 5: public parameters?

- "Traditional" approach to choosing **a** in LWE/LWR schemes:

  "Let **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 **a** in LWE/LWR schemes:

  "Let **a** *be a uniformly random. . . "*

- Before NewHope: *real-world* approach: generate fixed **a** once
- What if **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 **a** in LWE/LWR schemes:

  “Let **a** *be a uniformly random. . . ”*
- Before NewHope: *real-world* approach: generate fixed **a** once
- What if **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 **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 **a** in LWE/LWR schemes:

  *"Let **a** be a uniformly random. . . "*

- Before NewHope: *real-world* approach: generate fixed **a** once
- What if **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 **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 **a** every time
- Server can cache **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 error, 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

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
- **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 $\mathrm{H}(s, C)$ 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 $\mathrm{H}(s, C)$ 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)

(Exercise) Resources

- Overview NIST round-2 candidates: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/round-2-submissions
- Slides from 2nd NIST standardization conference: https://csrc.nist.gov/Events/2019/Second-PQC-Standardization-Conference
- NIST PQC Wiki (Florida Atlantic University): https://pqc-wiki.fau.edu