Post-quantum cryptography

Peter Schwabe

Radboud University, Nijmegen, The Netherlands

August 4, 2016

Noisebridge, San Francisco

"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

"Whether we can control the quantum states and all of that at the fundamental level has now been proven. The big killer is, at what point do we build a processor big enough thats it's faster than a classical computer?

That means moving away from small scale models to integrated processing devices and prototypes. That's the challenge, and that can be done, we anticipate, within the next decade."

—Michelle Simmons (UNSW), Jan. 2016

Why would cryptographers care?

Grover's algorithm (1996)

- Find preimages of a blackbox function in $O(\sqrt{N})$
- N is the size of the domain of the function

Why would cryptographers care?

Grover's algorithm (1996)

- Find preimages of a blackbox function in $O(\sqrt{N})$
- N is the size of the domain of the function
- Find n-bit symmetric keys in $2^{n/2}$ "operations"
- Find hash-function preimages in $2^{n/2}$

Why would cryptographers care?

Grover's algorithm (1996)

- Find preimages of a blackbox function in $O(\sqrt{N})$
- N is the size of the domain of the function
- Find n-bit symmetric keys in $2^{n/2}$ "operations"
- Find hash-function preimages in $2^{n/2}$
- Consequences: double key lengths (and hash lengths)

Why would cryptographers care?

Grover's algorithm (1996)

- Find preimages of a blackbox function in $O(\sqrt{N})$
- N is the size of the domain of the function
- Find n-bit symmetric keys in $2^{n/2}$ "operations"
- Find hash-function preimages in $2^{n/2}$
- Consequences: double key lengths (and hash lengths)

Shor's algorithm (1994)

- Factor integers in polynomial time
- Compute discrete logarithms in polynomial time

Why would cryptographers care?

Grover's algorithm (1996)

- Find preimages of a blackbox function in $O(\sqrt{N})$
- N is the size of the domain of the function
- Find n-bit symmetric keys in $2^{n/2}$ "operations"
- Find hash-function preimages in $2^{n/2}$
- Consequences: double key lengths (and hash lengths)

Shor's algorithm (1994)

- Factor integers in polynomial time
- Compute discrete logarithms in polynomial time
- Complete break of RSA, ElGamal, DSA, Diffie-Hellman
- Complete break of elliptic-curve variants (ECSDA, ECDH, ...)

Is public-key crypto dead?

Is public-key crypto dead?

Alternative, "post-quantum" PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)

Is public-key crypto dead?

Alternative, "post-quantum" PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)
- Code-based cryptography (e.g., McEliece encryption)

Is public-key crypto dead?

Alternative, "post-quantum" PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)
- Code-based cryptography (e.g., McEliece encryption)
- Multivariate signatures (e.g., UOV, HFEv-)

Is public-key crypto dead?

Alternative, "post-quantum" PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)
- Code-based cryptography (e.g., McEliece encryption)
- Multivariate signatures (e.g., UOV, HFEv-)
- Lattice-based crypto (e.g., NTRU, LWE encryption)

Is public-key crypto dead?

Alternative, "post-quantum" PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)
- Code-based cryptography (e.g., McEliece encryption)
- Multivariate signatures (e.g., UOV, HFEv-)
- Lattice-based crypto (e.g., NTRU, LWE encryption)
- Supersingular isogeny crypto (SIDH)

Is public-key crypto dead?

Alternative, “post-quantum” PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)
- Code-based cryptography (e.g., McEliece encryption)
- Multivariate signatures (e.g., UOV, HFEv-)
- Lattice-based crypto (e.g., NTRU, LWE encryption)
- Supersingular isogeny crypto (SIDH)

Why aren’t we using those?

- Slower computation (for some)

Is public-key crypto dead?

Alternative, "post-quantum" PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)
- Code-based cryptography (e.g., McEliece encryption)
- Multivariate signatures (e.g., UOV, HFEv-)
- Lattice-based crypto (e.g., NTRU, LWE encryption)
- Supersingular isogeny crypto (SIDH)

Why aren't we using those?

- Slower computation (for some)
- Larger keys, signatures, ciphertexts (for some)

Is public-key crypto dead?

Alternative, "post-quantum" PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)
- Code-based cryptography (e.g., McEliece encryption)
- Multivariate signatures (e.g., UOV, HFEv-)
- Lattice-based crypto (e.g., NTRU, LWE encryption)
- Supersingular isogeny crypto (SIDH)

Why aren't we using those?

- Slower computation (for some)
- Larger keys, signatures, ciphertexts (for some)
- Security less well understood (for some)

Is public-key crypto dead?

Alternative, "post-quantum" PKC

- Hash-based signatures (e.g., XMSS, SPHINCS)
- Code-based cryptography (e.g., McEliece encryption)
- Multivariate signatures (e.g., UOV, HFEv-)
- Lattice-based crypto (e.g., NTRU, LWE encryption)
- Supersingular isogeny crypto (SIDH)

Why aren't we using those?

- Slower computation (for some)
- Larger keys, signatures, ciphertexts (for some)
- Security less well understood (for some)
- Additional issues (e.g., stateful hash-based signing)

NIST post-quantum crypto project

- NIST issued a (draft) call for PQC proposals
- Submissions for
  - PQ signatures
  - PQ encryption
  - PQ key agreement
- Submission deadline: November 2017
- Submitters' presentations: Early 2018
- 3–5 years of analysis
- 2 years later: draft standards ready
- See `http://csrc.nist.gov/groups/ST/post-quantum-crypto/`

PQCRYPTO

- Project funded by EU in Horizon 2020.
- Starting date 1 March 2015, runs for 3 years.
- 11 partners from academia and industry, TU/e is coordinator
- Goal: **Design and implement high-security post-quantum PKC**

NSA's data center in Bluffdale

NSA's data center in Bluffdale

Estimated numbers

- Electricity consumption: 65 MW
- Energy bill: US$40,000,000$/year
- Storage: 3–12 EB

NSA's data center in Bluffdale

Estimated numbers

- Electricity consumption: 65 MW
- Energy bill: US$40,000,000$/year
- Storage: 3–12 EB

The attack scenario

- Store encrypted data now
- Decrypt in 15 (?) years

NSA's data center in Bluffdale

Estimated numbers

- Electricity consumption: 65 MW
- Energy bill: US$40,000,000$/year
- Storage: 3–12 EB

The attack scenario

- Store encrypted data now
- Decrypt in 15 $(?)$ years
- Consequence:

Need post-quantum encryption now!

How about PFS?

- "Perfect Forward Secrecy":
  - Use long-term secret keys for authentication only
  - Use short-term *ephemeral* keys for encryption
  - Compromise of long-term key does not compromise confidentiality of past messages

How about PFS?

- "Perfect Forward Secrecy":
  - Use long-term secret keys for authentication only
  - Use short-term *ephemeral* keys for encryption
  - Compromise of long-term key does not compromise confidentiality of past messages
- *Does not help* against cryptanalytic break
- Attacker breaks (in poly time) each single ephemeral key exchange

How about PFS?

- "Perfect Forward Secrecy":
  - Use long-term secret keys for authentication only
  - Use short-term *ephemeral* keys for encryption
  - Compromise of long-term key does not compromise confidentiality of past messages
- *Does not help* against cryptanalytic break
- Attacker breaks (in poly time) each single ephemeral key exchange
- As a consequence, we want
  - **ephemeral key exchange** (to protect against key compromise)
  - **post-quantum security** (to protect against future quantum attacker)

POST-QUANTUM KEY EXCHANGE

A NEW HOPE

ERDEM ALKIM
LÉO DUCAS
THOMAS PÖPPELMANN
PETER SCHWABE

Ring-Learning-with-errors (RLWE)

- Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Let χ be an *error distribution* on $\mathcal{R}_q$
- Let $\mathbf{s} \in \mathcal{R}_q$ be secret
- Attacker is given pairs $(\mathbf{a}, \mathbf{as} + \mathbf{e})$ with
  - $\mathbf{a}$ uniformly random from $\mathcal{R}_q$
  - $\mathbf{e}$ sampled from χ
- Task for the attacker: find $\mathbf{s}$

Ring-Learning-with-errors (RLWE)

- Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Let χ be an *error distribution* on $\mathcal{R}_q$
- Let $\mathbf{s} \in \mathcal{R}_q$ be secret
- Attacker is given pairs $(\mathbf{a}, \mathbf{as} + \mathbf{e})$ with
  - $\mathbf{a}$ uniformly random from $\mathcal{R}_q$
  - $\mathbf{e}$ sampled from χ
- Task for the attacker: find $\mathbf{s}$
- Common choice for χ: discrete Gaussian

Ring-Learning-with-errors (RLWE)

- Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Let χ be an *error distribution* on $\mathcal{R}_q$
- Let $\mathbf{s} \in \mathcal{R}_q$ be secret
- Attacker is given pairs $(\mathbf{a}, \mathbf{as} + \mathbf{e})$ with
  - $\mathbf{a}$ uniformly random from $\mathcal{R}_q$
  - $\mathbf{e}$ sampled from χ
- Task for the attacker: find $\mathbf{s}$
- Common choice for χ: discrete Gaussian
- Common optimization for protocols: fix $\mathbf{a}$

A bit of (R)LWE history

- Regev, 2005: Introduce LWE-based encryption
- Lyubashevsky, Peikert, Regev, 2010: Ring-LWE and Ring-LWE encryption
- Ding, Xie, Lin, 2012: Transform to (R)LWE-based key exchange
- Peikert, 2014: Improved RLWE-based key exchange
- Bos, Costello, Naehrig, Stebila, 2015: Instantiate and implement Peikert's KEX in TLS

Peikert's RLWE-based KEM

Parameters: q, n, χ		
KEM.Setup() :		
$\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$		
Alice (server)		Bob (client)
KEM.Gen($\mathbf{a}$) :		KEM.Encaps($\mathbf{a}, \mathbf{b}$) :
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{\mathbf{b}}$	$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$\bar{\mathbf{v}} \xleftarrow{\$} \mathsf{dbl}(\mathbf{v})$
KEM.Decaps($\mathbf{s}, (\mathbf{u}, \mathbf{v}')$) :	$\xleftarrow{\mathbf{u}, \mathbf{v}'}$	$\mathbf{v}' = \langle \bar{\mathbf{v}} \rangle_2$
$\mu \leftarrow \mathsf{rec}(2\mathbf{us}, \mathbf{v}')$		$\mu \leftarrow \lfloor \bar{\mathbf{v}} \rceil_2$

Peikert's RLWE-based KEM

Parameters: q, n, χ		
KEM.Setup() :		
$\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$		
Alice (server)		Bob (client)
KEM.Gen($\mathbf{a}$) :		KEM.Encaps($\mathbf{a}, \mathbf{b}$) :
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{\mathbf{b}}$	$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$\bar{\mathbf{v}} \xleftarrow{\$} \mathsf{dbl}(\mathbf{v})$
KEM.Decaps($\mathbf{s}, (\mathbf{u}, \mathbf{v}')$) :	$\xleftarrow{\mathbf{u}, \mathbf{v}'}$	$\mathbf{v}' = \langle \bar{\mathbf{v}} \rangle_2$
$\mu \leftarrow \mathsf{rec}(2\mathbf{us}, \mathbf{v}')$		$\mu \leftarrow \lfloor \bar{\mathbf{v}} \rceil_2$

Observe: $2\mathbf{us} = 2\mathbf{ass}' + 2\mathbf{e}'\mathbf{s} \approx 2\mathbf{ass}' + 2\mathbf{es}' + 2\mathbf{e}'' \approx \bar{\mathbf{v}}$

BCNS key exchange

- Bos, Costello, Naehrig, Stebila, IEEE S&P 2015:
  - Phrase the KEM as key exchange
  - Instantiate with concrete parameters
  - Integrate with OpenSSL $\rightarrow$ post-quantum TLS key exchange
  - Also: combined ECDH+RLWE key exchange

BCNS key exchange

- Bos, Costello, Naehrig, Stebila, IEEE S&P 2015:
  - Phrase the KEM as key exchange
  - Instantiate with concrete parameters
  - Integrate with OpenSSL $\to$ post-quantum TLS key exchange
  - Also: combined ECDH+RLWE key exchange
- Parameters chosen by BCNS:
  - $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
  - $n = 1024$
  - $q = 2^{32} - 1$
  - $\chi = D_{\mathbb{Z},\sigma}$
  - $\sigma = 8/\sqrt{2\pi} \approx 3.192$

BCNS key exchange

- Bos, Costello, Naehrig, Stebila, IEEE S&P 2015:
  - Phrase the KEM as key exchange
  - Instantiate with concrete parameters
  - Integrate with OpenSSL $\rightarrow$ post-quantum TLS key exchange
  - Also: combined ECDH+RLWE key exchange
- Parameters chosen by BCNS:
  - $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
  - $n = 1024$
  - $q = 2^{32} - 1$
  - $\chi = D_{\mathbb{Z},\sigma}$
  - $\sigma = 8/\sqrt{2\pi} \approx 3.192$
- Claimed security level: 128 bits pre-quantum
- Failure probability: $\approx 2^{-131072}$

A new hope

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$

A new hope

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Drastically reduce q to $12289 < 2^{14}$
- Still use $n = 1024$

A new hope

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Drastically reduce q to $12289 < 2^{14}$
- Still use $n = 1024$
- Analysis of *post-quantum* security

A new hope

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Drastically reduce q to $12289 < 2^{14}$
- Still use $n = 1024$
- Analysis of *post-quantum* security
- Use centered binomial noise ψ_k ($\sum_{i=1}^{k} b_i - b'_i$ for $b_i, b'_i \in \{0, 1\}$)

A new hope

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Drastically reduce q to $12289 < 2^{14}$
- Still use $n = 1024$
- Analysis of *post-quantum* security
- Use centered binomial noise ψ_k ($\sum_{i=1}^{k} b_i - b'_i$ for $b_i, b'_i \in \{0,1\}$)
- Choose a fresh parameter $\mathbf{a}$ for every protocol run

A new hope

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Drastically reduce q to $12289 < 2^{14}$
- Still use $n = 1024$
- Analysis of *post-quantum* security
- Use centered binomial noise ψ_k ($\sum_{i=1}^{k} b_i - b'_i$ for $b_i, b'_i \in \{0, 1\}$)
- Choose a fresh parameter $\mathbf{a}$ for every protocol run
- Encode polynomials in NTT domain

A new hope

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Drastically reduce q to $12289 < 2^{14}$
- Still use $n = 1024$
- Analysis of *post-quantum* security
- Use centered binomial noise ψ_k ($\sum_{i=1}^{k} b_i - b'_i$ for $b_i, b'_i \in \{0, 1\}$)
- Choose a fresh parameter $\mathbf{a}$ for every protocol run
- Encode polynomials in NTT domain
- Multiple implementations

A new hope – protocol

Parameters: $q = 12289 < 2^{14},\ n = 1024$

Error distribution: ψ_{16}

Alice (server)		Bob (client)
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \psi_{16}^n$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \psi_{16}^n$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{r})}$	$\mathbf{r} \xleftarrow{\$} \mathsf{HelpRec}(\mathbf{v})$
$k \leftarrow \mathsf{Rec}(\mathbf{v}', \mathbf{r})$		$k \leftarrow \mathsf{Rec}(\mathbf{v}, \mathbf{r})$
$\mu \leftarrow \mathsf{SHA3\text{-}256}(k)$		$\mu \leftarrow \mathsf{SHA3\text{-}256}(k)$

Error reconciliation

- After running the protocol
  - Alice has $\mathbf{x}_A = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}'\mathbf{s}$
  - Bob has $\mathbf{x}_B = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}\mathbf{s}' + \mathbf{e}''$
- Those elements are similar, but not the same
- Problem: How to agree on *the same* key from these noisy vectors?

Error reconciliation

- After running the protocol
  - Alice has $\mathbf{x}_A = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}'\mathbf{s}$
  - Bob has $\mathbf{x}_B = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}\mathbf{s}' + \mathbf{e}''$
- Those elements are similar, but not the same
- Problem: How to agree on *the same* key from these noisy vectors?
- Known: Extract one bit from each coefficient
- Also known: Extract multiple bits from each coefficient (decrease security)

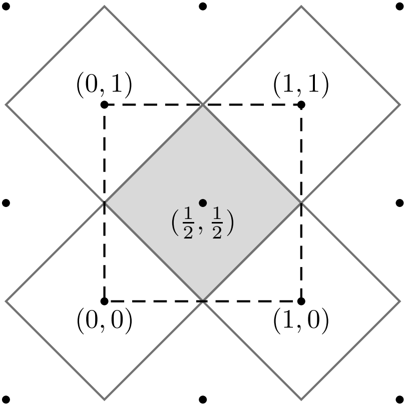
Error reconciliation

- After running the protocol
  - Alice has $\mathbf{x}_A = \mathbf{ass}' + \mathbf{e}'\mathbf{s}$
  - Bob has $\mathbf{x}_B = \mathbf{ass}' + \mathbf{es}' + \mathbf{e}''$
- Those elements are similar, but not the same
- Problem: How to agree on *the same* key from these noisy vectors?
- Known: Extract one bit from each coefficient
- Also known: Extract multiple bits from each coefficient (decrease security)
- NewHope: extract one bit from multiple coefficients (increase security)
- Specifically: 1 bit from 4 coefficients $\rightarrow$ 256-bit key from 1024 coefficients

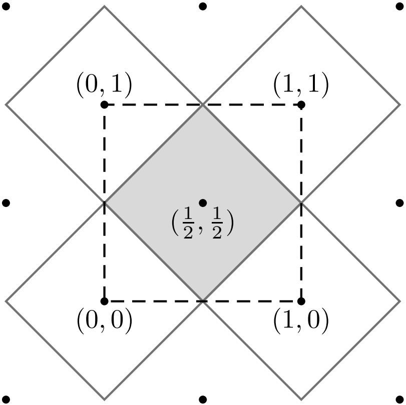
Error reconciliation

- After running the protocol
  - Alice has $\mathbf{x}_A = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}'\mathbf{s}$
  - Bob has $\mathbf{x}_B = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}\mathbf{s}' + \mathbf{e}''$
- Those elements are similar, but not the same
- Problem: How to agree on *the same* key from these noisy vectors?
- Known: Extract one bit from each coefficient
- Also known: Extract multiple bits from each coefficient (decrease security)
- NewHope: extract one bit from multiple coefficients (increase security)
- Specifically: 1 bit from 4 coefficients $\rightarrow$ 256-bit key from 1024 coefficients
- In the following: 2-dimensional intuition (4-dim. case very similar)
- "Scale" vector $\mathbf{x}$ to $[0, 1)^2$

2D Error reconciliation

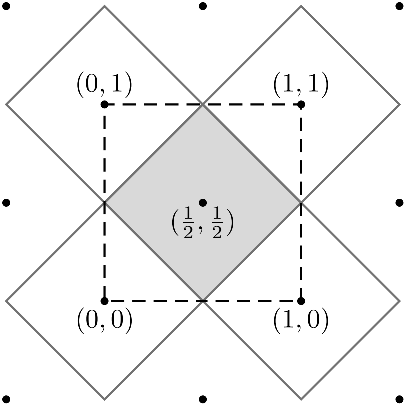


2D Error reconciliation



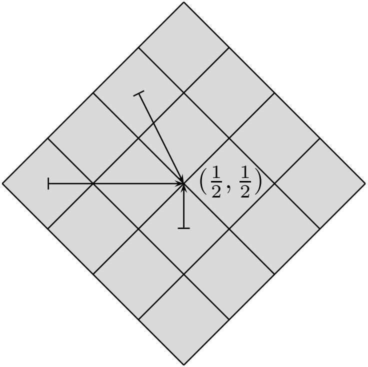
- If $\mathbf{x}$ is in the grey Voronoi cell: pick key bit 1
- If $\mathbf{x}$ is in the white Voronoi cell: pick key bit 0

2D Error reconciliation



- If $\mathbf{x}$ is in the grey Voronoi cell: pick key bit 1
- If $\mathbf{x}$ is in the white Voronoi cell: pick key bit 0
- Reconciliation: Bob sends difference vector from $\mathbf{x}_B$ to center of his Voronoi cell
- Alice adds this difference vector to her vector $\mathbf{x}_A$

Discretization of reconciliation



- Sending difference vector means doubling communcation
- Idea: chop Voronoi cell into 2^{dr} subcells
  - d: dimension (4 for NewHope, 2 in this picture)
  - r: discretization level
- Need to send only rd bits per d coefficients
- NewHope: $r = 2$; hence 256 bytes of reconciliation information

"Blurring the edges"

- This would all work if $\mathbf{x}$ was continuous uniform from $[0, 1)$
- We start with $\mathbf{x} \in \{0, \ldots, q-1\}^2$, q odd
- Odd number of possible values; no way to pick key bit without bias!
- This is the same for dimension 4

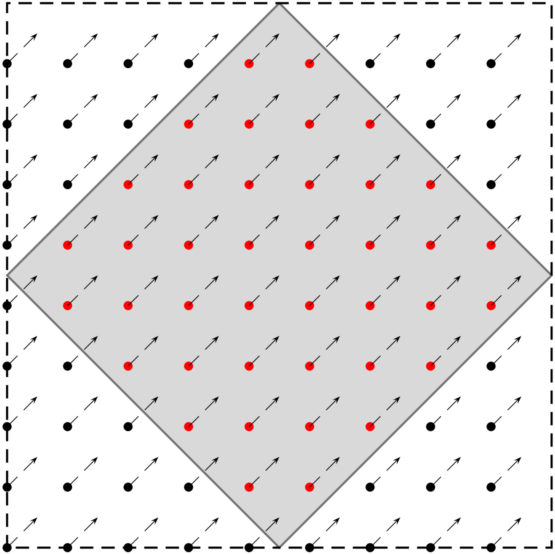
"Blurring the edges"

- This would all work if $\mathbf{x}$ was continuous uniform from $[0, 1)$
- We start with $\mathbf{x} \in \{0, \ldots, q-1\}^2$, q odd
- Odd number of possible values; no way to pick key bit without bias!
- This is the same for dimension 4
- Idea: randomly "blur the edges"
- Add vector $(1/2q, 1/2q)$ with probability $1/2$ before reconciliation
- This is a generalization of Peikert's "randomized doubling" trick

"Blurring the edges"

- This would all work if $\mathbf{x}$ was continuous uniform from $[0, 1)$
- We start with $\mathbf{x} \in \{0, \ldots, q-1\}^2$, q odd
- Odd number of possible values; no way to pick key bit without bias!
- This is the same for dimension 4
- Idea: randomly "blur the edges"
- Add vector $(1/2q, 1/2q)$ with probability $1/2$ before reconciliation
- This is a generalization of Peikert's "randomized doubling" trick

"Blurring the edges"



Security analysis

- Consider RLWE instance as LWE instance
- Attack using BKZ
- BKZ uses SVP oracle in smaller dimension
- Consider only the cost of one call to that oracle ("core-SVP hardness")

Security analysis

- Consider RLWE instance as LWE instance
- Attack using BKZ
- BKZ uses SVP oracle in smaller dimension
- Consider only the cost of one call to that oracle ("core-SVP hardness")
- Consider quantum sieve as SVP oracle
  - Best-known quantum cost (BKC): $2^{0.265n}$
  - Best-plausible quantum cost (BPC): $2^{0.2075n}$

Security analysis

- Consider RLWE instance as LWE instance
- Attack using BKZ
- BKZ uses SVP oracle in smaller dimension
- Consider only the cost of one call to that oracle ("core-SVP hardness")
- Consider quantum sieve as SVP oracle
  - Best-known quantum cost (BKC): $2^{0.265n}$
  - Best-plausible quantum cost (BPC): $2^{0.2075n}$
- Primal attack: unique-SVP from LWE; solve using BKZ

Security analysis

- Consider RLWE instance as LWE instance
- Attack using BKZ
- BKZ uses SVP oracle in smaller dimension
- Consider only the cost of one call to that oracle ("core-SVP hardness")
- Consider quantum sieve as SVP oracle
  - Best-known quantum cost (BKC): $2^{0.265n}$
  - Best-plausible quantum cost (BPC): $2^{0.2075n}$
- Primal attack: unique-SVP from LWE; solve using BKZ
- Dual attack: find short vector in dual lattice
- Length determines complexity and attacker's advantage ϵ

JarJar

"I don't like is the way that the parameters are set [...] I think that setting them too high impedes research."

—*anonymous reviewer*

JarJar

"I don't like is the way that the parameters are set [...] I think that setting them too high impedes research."

—anonymous reviewer

- JarJar: instantiation with $n = 512$
- Same $q = 12289$
- Use root lattice D_2 instead of D_4
- Use $k = 24$ for the centered binomial distribution

JarJar

"I don't like is the way that the parameters are set [. . .] I think that setting them too high impedes research."

—*anonymous reviewer*

- JarJar: instantiation with $n = 512$
- Same $q = 12289$
- Use root lattice D_2 instead of D_4
- Use $k = 24$ for the centered binomial distribution

JarJar is not recommended for use!

Post-quantum security

Attack	m	b	Known Classical	Known Quantum	Best Plausible
BCNS proposal: $q = 2^{32} - 1$, $n = 1024$, $\sigma = 3.192$					
Primal	1062	296	86	78	61
Dual	1055	296	86	**78**	61
JarJar: $q = 12289$, $n = 512$, $\sigma = \sqrt{12}$					
Primal	623	449	131	119	93
Dual	602	448	131	**118**	92
NewHope: $q = 12289$, $n = 1024$, $\sigma = \sqrt{8}$					
Primal	1100	967	282	256	200
Dual	1099	962	281	**255**	199

- b: Block size for BKZ
- m: Number of used samples

Against all authority

- Remember the optimization of fixed $\mathbf{a}$?
- What if $\mathbf{a}$ is backdoored?
- Parameter-generating authority can break key exchange
- "Solution": Nothing-up-my-sleeves (involves endless discussion!)

Against all authority

- Remember the optimization of fixed $\mathbf{a}$?
- 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

Against all authority

- Remember the optimization of fixed $\mathbf{a}$?
- 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
- Use SHAKE-128 to expand a 32-byte seed

Against all authority

- Remember the optimization of fixed $\mathbf{a}$?
- 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
- Use SHAKE-128 to expand a 32-byte seed
- Server can cache $\mathbf{a}$ for some time (e.g., 1h)

Against all authority

- Remember the optimization of fixed $\mathbf{a}$?
- 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
- Use SHAKE-128 to expand a 32-byte seed
- Server can cache $\mathbf{a}$ for some time (e.g., 1h)
- **Must not reuse keys/noise!**

NTT-based multiplication

- Most costly arithmetic operations: multiplication in $\mathcal{R}_q$
- Idea behind selecting n and q: fast negacyclic number-theoretic transform (NTT)
- This requires that $2n$ divides $q - 1$
- Note that $2n = 2^{11}$ divides $12288 = 2^{13} + 2^{12}$

NTT-based multiplication

- Most costly arithmetic operations: multiplication in $\mathcal{R}_q$
- Idea behind selecting n and q: fast negacyclic number-theoretic transform (NTT)
- This requires that $2n$ divides $q-1$
- Note that $2n = 2^{11}$ divides $12288 = 2^{13} + 2^{12}$
- To multiply $\mathbf{f}$ and $\mathbf{g}$ in $\mathcal{R}_q$:
  - Compute $\hat{\mathbf{f}} = \mathsf{NTT}(\mathbf{f})$
  - Compute $\hat{\mathbf{g}} = \mathsf{NTT}(\mathbf{g})$

NTT-based multiplication

- Most costly arithmetic operations: multiplication in $\mathcal{R}_q$
- Idea behind selecting n and q: fast negacyclic number-theoretic transform (NTT)
- This requires that $2n$ divides $q - 1$
- Note that $2n = 2^{11}$ divides $12288 = 2^{13} + 2^{12}$
- To multiply $\mathbf{f}$ and $\mathbf{g}$ in $\mathcal{R}_q$:
  - Compute $\hat{\mathbf{f}} = \mathsf{NTT}(\mathbf{f})$
  - Compute $\hat{\mathbf{g}} = \mathsf{NTT}(\mathbf{g})$
  - $\hat{\mathbf{f}}$ and $\hat{\mathbf{g}}$ have 1024 coefficients each
  - Multiply componentwise to obtain $\hat{\mathbf{r}}$

NTT-based multiplication

- Most costly arithmetic operations: multiplication in $\mathcal{R}_q$
- Idea behind selecting n and q: fast negacyclic number-theoretic transform (NTT)
- This requires that $2n$ divides $q-1$
- Note that $2n = 2^{11}$ divides $12288 = 2^{13} + 2^{12}$
- To multiply $\mathbf{f}$ and $\mathbf{g}$ in $\mathcal{R}_q$:
  - Compute $\hat{\mathbf{f}} = \mathsf{NTT}(\mathbf{f})$
  - Compute $\hat{\mathbf{g}} = \mathsf{NTT}(\mathbf{g})$
  - $\hat{\mathbf{f}}$ and $\hat{\mathbf{g}}$ have 1024 coefficients each
  - Multiply componentwise to obtain $\hat{\mathbf{r}}$
  - Compute result of multiplication as $\mathbf{r} = \mathsf{NTT}^{-1}(\mathbf{r})$

NTT-based multiplication

- Most costly arithmetic operations: multiplication in $\mathcal{R}_q$
- Idea behind selecting n and q: fast negacyclic number-theoretic transform (NTT)
- This requires that $2n$ divides $q - 1$
- Note that $2n = 2^{11}$ divides $12288 = 2^{13} + 2^{12}$
- To multiply $\mathbf{f}$ and $\mathbf{g}$ in $\mathcal{R}_q$:
  - Compute $\hat{\mathbf{f}} = \mathsf{NTT}(\mathbf{f})$
  - Compute $\hat{\mathbf{g}} = \mathsf{NTT}(\mathbf{g})$
  - $\hat{\mathbf{f}}$ and $\hat{\mathbf{g}}$ have 1024 coefficients each
  - Multiply componentwise to obtain $\hat{\mathbf{r}}$
  - Compute result of multiplication as $\mathbf{r} = \mathsf{NTT}^{-1}(\mathbf{r})$
- NTT takes $\frac{n}{2}\log(n)$ "butterfly operations"
- Butterflies are one addition, one subtraction, one multiplication by constant

Implementation

- Very fast multiplication in $\mathcal{R}_q$: use NTT
- Define message format:
  - Send polynomials in NTT domain
  - Eliminate two of the required NTTs

Implementation

- Very fast multiplication in $\mathcal{R}_q$: use NTT
- Define message format:
  - Send polynomials in NTT domain
  - Eliminate two of the required NTTs
- C reference implementation:
  - Arithmetic on 16-bit and 32-bit integers
  - No division (/) or modulo (%) operator
  - Use Montgomery reductions inside NTT
  - Use ChaCha20 for noise sampling

Implementation

- Very fast multiplication in $\mathcal{R}_q$: use NTT
- Define message format:
  - Send polynomials in NTT domain
  - Eliminate two of the required NTTs
- C reference implementation:
  - Arithmetic on 16-bit and 32-bit integers
  - No division (/) or modulo (%) operator
  - Use Montgomery reductions inside NTT
  - Use ChaCha20 for noise sampling
- AVX2 implementation:
  - Speed up NTT using vectorized `double` arithmetic
  - Use AES-256 for noise sampling
  - Use AVX2 for centered binomial

The protocol revisited

Parameters: $q = 12289 < 2^{14}$, $n = 1024$

Error distribution: ψ_{16}^n

Alice (server)		**Bob (client)**
$seed \xleftarrow{\$} \{0,\ldots,255\}^{32}$		
$\hat{\mathbf{a}} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \psi_{16}^n$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \psi_{16}^n$
$\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$		
$\hat{\mathbf{b}} \leftarrow \hat{\mathbf{a}} \circ \hat{\mathbf{s}} + \mathsf{NTT}(\mathbf{e})$	$\xrightarrow[\text{1824 Bytes}]{m_a = \mathsf{encodeA}(seed, \hat{\mathbf{b}})}$	$(\hat{\mathbf{b}}, seed) \leftarrow \mathsf{decodeA}(m_a)$
		$\hat{\mathbf{a}} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$
		$\hat{\mathbf{t}} \leftarrow \mathsf{NTT}(\mathbf{s}')$
		$\hat{\mathbf{u}} \leftarrow \hat{\mathbf{a}} \circ \hat{\mathbf{t}} + \mathsf{NTT}(\mathbf{e}')$
		$\mathbf{v} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{b}} \circ \hat{\mathbf{t}}) + \mathbf{e}''$
$(\hat{\mathbf{u}}, \mathbf{r}) \leftarrow \mathsf{decodeB}(m_b)$	$\xleftarrow[\text{2048 Bytes}]{m_b = \mathsf{encodeB}(\hat{\mathbf{u}}, \mathbf{r})}$	$\mathbf{r} \xleftarrow{\$} \mathsf{HelpRec}(\mathbf{v})$
$\mathbf{v}' \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{u}} \circ \hat{\mathbf{s}})$		$k \leftarrow \mathsf{Rec}(\mathbf{v}, \mathbf{r})$
$k \leftarrow \mathsf{Rec}(\mathbf{v}', \mathbf{r})$		$\mu \leftarrow \mathsf{SHA3\text{-}256}(k)$
$\mu \leftarrow \mathsf{SHA3\text{-}256}(k)$		

Performance

	BCNS	C ref	AVX2
Key generation (server)	$\approx 2\,477\,958$	$258\,246$ ($258\,965$)	$88\,920$ ($89\,079$)
Key gen + shared key (client)	$\approx 3\,995\,977$	$384\,994$ ($385\,146$)	$110\,986$ ($111\,169$)
Shared key (server)	$\approx 481\,937$	$86\,280$	$19\,422$

- Benchmarks on one core of an Intel i7-4770K (Haswell)
- BCNS benchmarks are derived from `openssl speed`
- Numbers in parantheses are average; all other numbers are median.
- Includes around $\approx 37\,000$ cycles for generation of $\mathbf{a}$ on each side

Performance

	BCNS	C ref	AVX2
Key generation (server)	$\approx 2\,477\,958$	$258\,246$ $(258\,965)$	$88\,920$ $(89\,079)$
Key gen + shared key (client)	$\approx 3\,995\,977$	$384\,994$ $(385\,146)$	$110\,986$ $(111\,169)$
Shared key (server)	$\approx 481\,937$	$86\,280$	$19\,422$

- Benchmarks on one core of an Intel i7-4770K (Haswell)
- BCNS benchmarks are derived from `openssl speed`
- Numbers in parantheses are average; all other numbers are median.
- Includes around $\approx 37\,000$ cycles for generation of $\mathbf{a}$ on each side
- Faster than state-of-the art ECC

NewHope on ARM Cortex M

- Joint work with Erdem Alkim and Philipp Jakubeit
- Optimize NewHope on Cortex M0 and M4
- 32-bit state-of-the art microcontrollers

NewHope on ARM Cortex M

- Joint work with Erdem Alkim and Philipp Jakubeit
- Optimize NewHope on Cortex M0 and M4
- 32-bit state-of-the art microcontrollers
- Start with C reference implementation
- New speed records for NTT from:
  - Montgomery reductions after constant multiplications
  - "Short Barrett reductions" after additions
  - Lazy reductions
  - Serious hand optimization in assembly

NewHope on ARM Cortex M

- Joint work with Erdem Alkim and Philipp Jakubeit
- Optimize NewHope on Cortex M0 and M4
- 32-bit state-of-the art microcontrollers
- Start with C reference implementation
- New speed records for NTT from:
  - Montgomery reductions after constant multiplications
  - "Short Barrett reductions" after additions
  - Lazy reductions
  - Serious hand optimization in assembly
- Also optimize other building blocks of NewHope

ARM Cortex-M results

- Server side: ≈ 1.47Mio cycles (M0) and $\approx 860\,000$ cycles (M4)
- Client side: ≈ 1.74Mio cycles (M0) and $\approx 985\,000$ cycles (M4)

ARM Cortex-M results

- Server side: ≈ 1.47Mio cycles (M0) and $\approx 860\,000$ cycles (M4)
- Client side: ≈ 1.74Mio cycles (M0) and $\approx 985\,000$ cycles (M4)
- Comparison to ECC: ≈ 3.59 cycles for X25519 scalar mult on M0

ARM Cortex-M results

- Server side: ≈ 1.47Mio cycles (M0) and $\approx 860\,000$ cycles (M4)
- Client side: ≈ 1.74Mio cycles (M0) and $\approx 985\,000$ cycles (M4)
- Comparison to ECC: ≈ 3.59 cycles for X25519 scalar mult on M0
- Comparison to HECC: ≈ 2.63 cycles on Kummer surface on M0

Should you use NewHope?

Should you use NewHope?

Yes, if. . .

- . . . you need post-quantum *ephemeral* key exchange *now*

Should you use NewHope?

Yes, if...

- ... you need post-quantum *ephemeral* key exchange *now*
- ... you combine it with (pre-quantum) ECDH (e.g., X25519)
  - Run both key exchanges, extract key from both shared keys
  - Be careful with extraction and authentication

Should you use NewHope?

Yes, if. . .

- . . . you need post-quantum *ephemeral* key exchange *now*
- . . . you combine it with (pre-quantum) ECDH (e.g., X25519)
  - Run both key exchanges, extract key from both shared keys
  - Be careful with extraction and authentication
- . . . you make sure that you can easily upgrade

Using NewHope

NewHope in TLS

- Google is running a post-quantum experiment
- Combination of NewHope and X25519 (called CECPQ1)
- Some connections from Chrome Canary to some Google services
- `https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html`

Using NewHope

NewHope in TLS

- Google is running a post-quantum experiment
- Combination of NewHope and X25519 (called CECPQ1)
- Some connections from Chrome Canary to some Google services
- `https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html`

NewHope in Tor?

- Proposal by Lovecruft and Schwabe: RebelAlliance
- Use NewHope and X25519 in Tor

Using NewHope

NewHope in TLS

- Google is running a post-quantum experiment
- Combination of NewHope and X25519 (called CECPQ1)
- Some connections from Chrome Canary to some Google services
- `https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html`

NewHope in Tor?

- Proposal by Lovecruft and Schwabe: RebelAlliance
- Use NewHope and X25519 in Tor
- Similar proposal for NTRU in Tor by Schanck, Whyte, and Zhang
- See paper from PETS 2016: `http://eprint.iacr.org/2015/287`

Using NewHope

NewHope in TLS

- Google is running a post-quantum experiment
- Combination of NewHope and X25519 (called CECPQ1)
- Some connections from Chrome Canary to some Google services
- https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html

NewHope in Tor?

- Proposal by Lovecruft and Schwabe: RebelAlliance
- Use NewHope and X25519 in Tor
- Similar proposal for NTRU in Tor by Schanck, Whyte, and Zhang
- See paper from PETS 2016: http://eprint.iacr.org/2015/287
- Plan: Merge these proposals

Future directions

- Try error-correcting codes for reconciliation?

Future directions

- Try error-correcting codes for reconciliation?
- Send polynomials in "normal" domain?
  - Decouple protocol from multiplication algorithm
  - Possibly drop least significant bits

Future directions

- Try error-correcting codes for reconciliation?
- Send polynomials in “normal” domain?
  - Decouple protocol from multiplication algorithm
  - Possibly drop least significant bits
- Use smaller q?
  - Smaller messages
  - Higher security
  - Does not support efficient negacyclic NTT

Future directions

- Try error-correcting codes for reconciliation?
- Send polynomials in "normal" domain?
  - Decouple protocol from multiplication algorithm
  - Possibly drop least significant bits
- Use smaller q?
  - Smaller messages
  - Higher security
  - Does not support efficient negacyclic NTT
- How about Nussbaumer's algorithm for multiplication?

Future directions

- Try error-correcting codes for reconciliation?
- Send polynomials in “normal” domain?
  - Decouple protocol from multiplication algorithm
  - Possibly drop least significant bits
- Use smaller q?
  - Smaller messages
  - Higher security
  - Does not support efficient negacyclic NTT
- How about Nussbaumer’s algorithm for multiplication?
- How about Karatsuba $+$ Toom for multiplication?

Future directions

- Try error-correcting codes for reconciliation?
- Send polynomials in "normal" domain?
  - Decouple protocol from multiplication algorithm
  - Possibly drop least significant bits
- Use smaller q?
  - Smaller messages
  - Higher security
  - Does not support efficient negacyclic NTT
- How about Nussbaumer's algorithm for multiplication?
- How about Karatsuba + Toom for multiplication?
- How about smaller n (e.g., $n \approx 800$)?

Future directions ctd.

- Authenticated key exchange (AKE):
  - Paper by Zhang, Zhang, Ding, Snook, Dagdelen, 2015: about $100\times$ slower than NewHope
  - Can we do better?

Future directions ctd.

- Authenticated key exchange (AKE):
  - Paper by Zhang, Zhang, Ding, Snook, Dagdelen, 2015: about $100\times$ slower than NewHope
  - Can we do better?
- How about Frodo?
  - Paper by Bos, Costello, Ducas, Mironov, Naehrig, Nikolaenko, Raghunathan, Stebila
  - See http://eprint.iacr.org/2016/659

Future directions ctd.

- Authenticated key exchange (AKE):
  - Paper by Zhang, Zhang, Ding, Snook, Dagdelen, 2015: about $100\times$ slower than NewHope
  - Can we do better?
- How about Frodo?
  - Paper by Bos, Costello, Ducas, Mironov, Naehrig, Nikolaenko, Raghunathan, Stebila
  - See http://eprint.iacr.org/2016/659
- How about NTRU-based key exchange?
  - Performance looks worse for ephemeral key exchange
  - How about authenticated key exchange?

Future directions ctd.

- Authenticated key exchange (AKE):
  - Paper by Zhang, Zhang, Ding, Snook, Dagdelen, 2015: about $100\times$ slower than NewHope
  - Can we do better?
- How about Frodo?
  - Paper by Bos, Costello, Ducas, Mironov, Naehrig, Nikolaenko, Raghunathan, Stebila
  - See `http://eprint.iacr.org/2016/659`
- How about NTRU-based key exchange?
  - Performance looks worse for ephemeral key exchange
  - How about authenticated key exchange?
- How about NTRU Prime?
  - Paper by Bernstein, Chuengsatiansup, Lange, van Vredendaal
  - See `http://eprint.iacr.org/2016/461`
  - Useful for ephemeral key exchange?

NewHope online

Paper:	https://cryptojedi.org/papers/#newhope
Software:	https://cryptojedi.org/crypto/#newhope
ARM Paper:	https://cryptojedi.org/papers/#newhopearm
ARM software:	https://github.com/newhopearm/newhopearm.git
Newhope in Go:	https://github.com/Yawning/newhope (by Yawning Angel)
Newhope in Rust:	https://code.ciph.re/isis/newhopers (by Isis Lovecruft)
Newhope in Java:	https://github.com/rweather/newhope-java (by Rhys Weatherley)
Newhope in Erlang:	https://github.com/ahf/luke (by Alexander Færøy)