# POST-QUANTUM KEY EXCHANGE

## A NEW HOPE

ERDEM ALKIM

LÉO DUCAS

THOMAS PÖPPELMANN

PETER SCHWABE

*"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

# The end of crypto as we know it

## 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, ...)

# The end of crypto as we know it

## 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, ...)

## Forward-secure post-quantum crypto

- Threatening *today*:
  - Attacker records encrypted messages now
  - Uses quantum computer in $1$-$2$ decades to break encryption

# The end of crypto as we know it

## 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, ...)

## Forward-secure post-quantum crypto

- Threatening *today*:
  - Attacker records encrypted messages now
  - Uses quantum computer in $1$-$2$ decades to break encryption
- "Perfect forward secrecy" (PFS) does not help
  - Countermeasure against key compromise
  - Not a countermeasure against cryptographic break

# The end of crypto as we know it

## 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, ...)

## Forward-secure post-quantum crypto

- Threatening *today*:
  - Attacker records encrypted messages now
  - Uses quantum computer in $1$-$2$ decades to break encryption
- "Perfect forward secrecy" (PFS) does not help
  - Countermeasure against key compromise
  - Not a countermeasure against cryptographic break
- Consequence: **Want post-quantum PFS crypto today**

# Ring-Learning-with-errors (RLWE)

- Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Let $\chi$ 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 $\chi$
- Task for the attacker: find $\mathbf{s}$

# Ring-Learning-with-errors (RLWE)

- Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Let $\chi$ 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 $\chi$
- Task for the attacker: find $\mathbf{s}$
- Common choice for $\chi$: discrete Gaussian
- Common optimization for protocols: fix $\mathbf{a}$

# A bit of (R)LWE history

- Hoffstein, Pipher, Silverman, 1996: NTRU cryptosystem
- 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 key exchange in TLS:

# A bit of (R)LWE history

- Hoffstein, Pipher, Silverman, 1996: NTRU cryptosystem
- 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 key exchange in TLS:
  - $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
  - $n = 1024$
  - $q = 2^{32} - 1$
  - $\chi = D_{\mathbb{Z},\sigma}$ (Discrete Gaussian) with $\sigma = 8/\sqrt{2\pi} \approx 3.192$

# A bit of (R)LWE history

- Hoffstein, Pipher, Silverman, 1996: NTRU cryptosystem
- 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 key exchange in TLS:
  - $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
  - $n = 1024$
  - $q = 2^{32} - 1$
  - $\chi = D_{\mathbb{Z},\sigma}$ (Discrete Gaussian) with $\sigma = 8/\sqrt{2\pi} \approx 3.192$
  - Claimed security level: $128$ bits pre-quantum
  - Failure probability: $\approx 2^{-131072}$

# BCNS key exchange

| | | |
|---|---|---|
| Parameters: $q = 2^{32} - 1, n = 1024$ | | |
| Error distribution: $\chi = D_{\mathbb{Z},\sigma}, \sigma = 8/\sqrt{2\pi}$ | | |
| Global system parameter: $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$ | | |
| Alice (server) | | Bob (client) |
| $\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$ | | $\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$ |
| $\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$ | $\xrightarrow{\mathbf{b}}$ | $\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$ |
| | | $\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$ |
| | | $\bar{\mathbf{v}} \xleftarrow{\$} \mathsf{dbl}(\mathbf{v})$ |
| | $\xleftarrow{\mathbf{u},\mathbf{v}'}$ | $\mathbf{v}' = \langle \bar{\mathbf{v}} \rangle_2$ |
| $\mu \leftarrow \mathsf{rec}(2\mathbf{us}, \mathbf{v}')$ | | $\mu \leftarrow \lfloor \bar{\mathbf{v}} \rceil_2$ |

Alice has $2\mathbf{us} = 2\mathbf{ass}' + 2\mathbf{e}'\mathbf{s}$

Bob has $\bar{\mathbf{v}} \approx 2\mathbf{v} = 2(\mathbf{bs}' + \mathbf{e}'') = 2((\mathbf{as} + \mathbf{e})\mathbf{s}' + \mathbf{e}'') = 2\mathbf{ass}' + 2\mathbf{es}' + 2\mathbf{e}''$

# A new hope

## Our contributions

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

# A new hope

## Our contributions

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Keep dimension $n = 1024$
- Drastically reduce $q$ to $12289 < 2^{14}$
- Higher security, shorter messages, and speedups

# A new hope

## Our contributions

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Keep dimension $n = 1024$
- Drastically reduce $q$ to $12289 < 2^{14}$
- Higher security, shorter messages, and speedups
- Analysis of *post-quantum* security

# A new hope

## Our contributions

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Keep dimension $n = 1024$
- Drastically reduce $q$ to $12289 < 2^{14}$
- Higher security, shorter messages, and speedups
- Analysis of *post-quantum* security
- Use centered binomial noise $\psi_k$ (HW($a$)$-$HW($b$) for $k$-bit $a, b$)

# A new hope

## Our contributions

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Keep dimension $n = 1024$
- Drastically reduce $q$ to $12289 < 2^{14}$
- Higher security, shorter messages, and speedups
- Analysis of *post-quantum* security
- Use centered binomial noise $\psi_k$ (HW($a$)−HW($b$) for $k$-bit $a, b$)
- Choose a fresh parameter $\mathbf{a}$ for every protocol run

# A new hope

## Our contributions

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Keep dimension $n = 1024$
- Drastically reduce $q$ to $12289 < 2^{14}$
- Higher security, shorter messages, and speedups
- Analysis of *post-quantum* security
- Use centered binomial noise $\psi_k$ (HW($a$)−HW($b$) for $k$-bit $a, b$)
- Choose a fresh parameter $\mathbf{a}$ for every protocol run
- Encode polynomials in NTT domain

# A new hope

## Our contributions

- Improve failure analysis and error reconciliation
- Choose parameters for failure probability $\approx 2^{-60}$
- Keep dimension $n = 1024$
- Drastically reduce $q$ to $12289 < 2^{14}$
- Higher security, shorter messages, and speedups
- Analysis of *post-quantum* security
- Use centered binomial noise $\psi_k$ (HW($a$)−HW($b$) for $k$-bit $a, b$)
- 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: $\psi_{16}$

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

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

Bob has $\mathbf{v} = \mathbf{bs}' + \mathbf{e}'' = (\mathbf{as} + \mathbf{e})\mathbf{s}' + \mathbf{e}'' = \mathbf{ass}' + \mathbf{es}' + \mathbf{e}''$

# 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; method inspired by analog error-correcting codes

# 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; method inspired by analog error-correcting codes
- Generalize Peikert's approach to obtain unbiased keys

# Post-quantum security

- 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”)

# Post-quantum security

- 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): $2^{0.2075n}$

# Post-quantum security

- 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): $2^{0.2075n}$
- Obtain lower bounds on the bit security:

| | Known Classical | Known Quantum | Best Plausible |
|---|---|---|---|
| BCNS | 86 | 78 | 61 |
| NewHope | 281 | 255 | 199 |

# 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!**

# Implementation

- Multiplication in $\mathcal{R}_q$ using number-theoretic transform (NTT)
- Message format:
  - Send polynomials in NTT domain
  - Eliminate two of the required NTTs

# Implementation

- Multiplication in $\mathcal{R}_q$ using number-theoretic transform (NTT)
- 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

- Multiplication in $\mathcal{R}_q$ using number-theoretic transform (NTT)
- 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 AVX2 for centered binomial
  - Use AVX2 for error reconciliation
  - Use AES-256 for noise sampling

# Performance

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

- Cycle counts from one core of an Intel i7-4770K (Haswell)
- BCNS benchmarks are derived from `openssl speed`
- Includes around $\approx 37\,000$ cycles for generation of $\mathbf{a}$ on each side
- Compare to X25519 elliptic-curve scalar mult: $156\,092$ cycles

# NewHope in the real world

- July 7, Google announces 2-year post-quantum experiment
- NewHope+X25519 (CECPQ1) in BoringSSL for Chrome Canary
- Used in access to select Google services

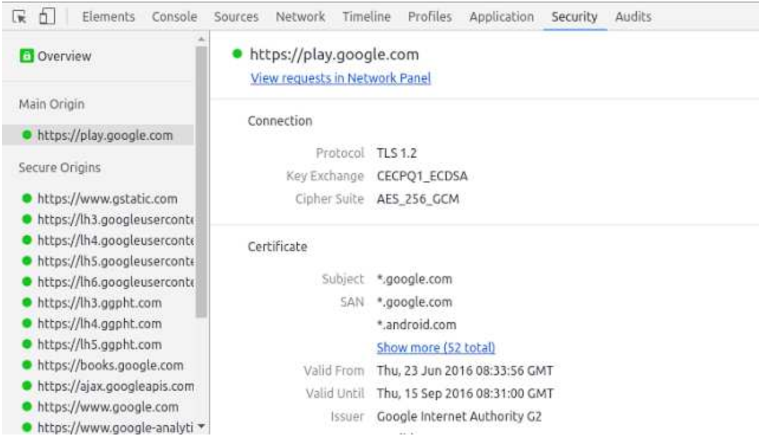


**Image source:** https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html

# NewHope online

| | |
|---|---|
| Paper: | https://cryptojedi.org/papers/#newhope |
| Software: | https://cryptojedi.org/crypto/#newhope |

# NewHope online

| | |
|---|---|
| Paper: | https://cryptojedi.org/papers/#newhope |
| Software: | https://cryptojedi.org/crypto/#newhope |
| Newhope for ARM: | https://github.com/newhopearm/newhopearm.git<br>(by Erdem Alkim, Philipp Jakubeit, and Peter Schwabe) |
| Newhope in Go: | https://github.com/Yawning/newhope<br>(by Yawning Angel) |
| Newhope in Rust: | https://code.ciph.re/isis/newhopers<br>(by Isis Lovecruft) |
| Newhope in Java: | https://github.com/rweather/newhope-java<br>(by Rhys Weatherley) |
| Newhope in Erlang: | https://github.com/ahf/luke<br>(by Alexander Færøy) |

newhope@cryptojedi.org