High-speed key encapsulation from NTRU

Andreas Hülsing[1], **Joost Rijneveld**[2], John Schanck[3,4], Peter Schwabe[2]

[1] Eindhoven University of Technology, The Netherlands
[2] Radboud University, Nijmegen, The Netherlands
[3] Institute for Quantum Computing, University of Waterloo, Canada
[4] Security Innovation, Wilmington, MA, USA

2017-09-26

CHES 2017

Post-quantum key exchange

Want to securely exchange a key ..

Post-quantum key exchange

Want to securely exchange a key ..

.. while the adversary has a quantum computer

Post-quantum key exchange

Want to securely exchange a key ..
.. while the adversary has a quantum computer

- Lattice-based schemes seem most promising
  - High speed, reasonable size
- Many schemes proposed, e.g.:
  [BCNS15], NewHope [ADPS16], Frodo [BCD+16], Lizard [CKLS16], Streamlined NTRU Prime [BCLvV17], spLWE-KEM [CHK+17], Kyber [BDK+17]
  - Typically with real-world parameters and implementations

Post-quantum key exchange

Want to securely exchange a key ..
.. while the adversary has a quantum computer

- Lattice-based schemes seem most promising
  - High speed, reasonable size
- Many schemes proposed, e.g.:
  [BCNS15], NewHope [ADPS16], Frodo [BCD+16], Lizard [CKLS16], Streamlined NTRU Prime [BCLvV17], spLWE-KEM [CHK+17], Kyber [BDK+17]
  - Typically with real-world parameters and implementations

This talk: back to the basics. NTRU [HPS98]

- Now without NTRUEncrypt patents!
- Faster & more secure parameters

This talk

- Describe parameter choices (and KEM)
  - Modulo some hand-waving
- Discuss implementation

This talk

- Describe parameter choices (and KEM)
  - Modulo some hand-waving
- Discuss implementation
  - Polynomial multiplications
  - Polynomial inversions
  - *Show that it can be fast and constant time*

This talk

- Describe parameter choices (and KEM)
  - Modulo some hand-waving
- Discuss implementation
  - Polynomial multiplications
  - Polynomial inversions
  - *Show that it can be fast and constant time*

Not this talk (see the paper!):

- Fast and constant time sampling routine
- History of NTRU
- Security analysis of parameters
- Discussion of alternatives
  - Ring-LWE, NTRU Prime, ..
- OW-CPA to OW-CCA2 transform [Den03] *in QROM*
  - 'Fusijaki-Okamoto transform for KEMs'

NTRU & parameters

- Three parameters: prime n, coprime integers p and q

NTRU & parameters

- Three parameters: prime n, coprime integers p and q
  - $n = 701$, $p = 3$, $q = 8192$

NTRU & parameters

- Three parameters: prime n, coprime integers p and q
  - $n = 701$, $p = 3$, $q = 8192$
- Define $R = \mathbb{Z}[x]/(x^n - 1)$ (i.e. polys of deg. n)

NTRU & parameters

- Three parameters: prime n, coprime integers p and q
  - $n = 701$, $p = 3$, $q = 8192$
- Define $R = \mathbb{Z}[x]/(x^n - 1)$ (i.e. polys of deg. n)
- Define $S = \mathbb{Z}[x]/\Phi_n$ (i.e. polys of deg. n-1)
  - $\Phi_n = x^{n-1} + \ldots + x^2 + x + 1$
  - $x^n - 1 = (x - 1) \cdot \Phi_n$

NTRU & parameters

- Three parameters: prime n, coprime integers p and q
  - $n = 701$, $p = 3$, $q = 8192$
- Define $R = \mathbb{Z}[x]/(x^n - 1)$ (i.e. polys of deg. n)
- Define $S = \mathbb{Z}[x]/\Phi_n$ (i.e. polys of deg. n-1)
  - $\Phi_n = x^{n-1} + \ldots + x^2 + x + 1$
  - $x^n - 1 = (x - 1) \cdot \Phi_n$

- sample $f, g \in S/3$ (i.e. coeffs. mod 3)
- lift f and g to f and g in R/q (i.e. coeffs. mod 8192)
- **Private key**: f
- **Public key**: $h = f^{-1} \cdot g \cdot (x - 1)$

NTRU & parameters

- Three parameters: prime n, coprime integers p and q
  - $n = 701$, $p = 3$, $q = 8192$
- Define $R = \mathbb{Z}[x]/(x^n - 1)$ (i.e. polys of deg. n)
- Define $S = \mathbb{Z}[x]/\Phi_n$ (i.e. polys of deg. n-1)
  - $\Phi_n = x^{n-1} + \ldots + x^2 + x + 1$
  - $x^n - 1 = (x - 1) \cdot \Phi_n$

- sample $f, g \in S/3$ (i.e. coeffs. mod 3)
- lift f and g to f and g in R/q (i.e. coeffs. mod 8192)
- **Private key**: f
- **Public key**: $h = f^{-1} \cdot g \cdot (x - 1)$
- **Encrypt**: $e = 3 \cdot r \cdot h + \text{lift}(m)$
- **Decrypt**: $m' = e \cdot f \cdot f^{-1}$ (reduce $R/q \to S/3$)

Parameter choices

- $n = 701$, $p = 3$, and $q = 8192$
- $R = \mathbb{Z}[x]/(x^n - 1)$, and $S = \mathbb{Z}[x]/\Phi_n$

- No decryption failures
  - Mild assumptions[1] on distribution for f, g
  - No assumptions on distribution for r, m

[1] Must be 'non-negatively correlated'; can be fast and constant time

Parameter choices

- $n = 701$, $p = 3$, and $q = 8192$
- $R = \mathbb{Z}[x]/(x^n - 1)$, and $S = \mathbb{Z}[x]/\Phi_n$

- No decryption failures
  - Mild assumptions[1] on distribution for f, g
  - No assumptions on distribution for r, m

- $\Phi_1 = (x - 1)$ as factor of h
  $\Rightarrow h \equiv 0 \bmod (q, \Phi_1)$
  $\Rightarrow$ No need for fixed Hamming-weight f and g
  $\Rightarrow$ No sorting or rejection sampling

[1] Must be 'non-negatively correlated'; can be fast and constant time

Parameter choices

- $n = 701$, $p = 3$, and $q = 8192$
- $R = \mathbb{Z}[x]/(x^n - 1)$, and $S = \mathbb{Z}[x]/\Phi_n$

- No decryption failures
  - Mild assumptions[1] on distribution for f, g
  - No assumptions on distribution for r, m

- $\Phi_1 = (x - 1)$ as factor of h
  $\Rightarrow h \equiv 0 \bmod (q, \Phi_1)$
  $\Rightarrow$ No need for fixed Hamming-weight f and g
  $\Rightarrow$ No sorting or rejection sampling

- Φ_{701} irreducible modulo 3 and q
  $\Rightarrow$ Every candidate f is invertible
  $\Rightarrow$ Easier constant time

[1] Must be 'non-negatively correlated'; can be fast and constant time

NTRU KEM

Transform OW-CPA to OW-CCA2 [Den03], *in QROM*

NTRU KEM

Transform OW-CPA to OW-CCA2 [Den03], *in QROM*

- Generate NTRU keypair
- Encapsulate:
  1. **Encrypt** *m* to randomized ciphertext
- Decapsulate:
  1. **Decrypt** to obtain *m*
  2. Re-**encrypt** *m* to verify correctness

NTRU KEM

Transform OW-CPA to OW-CCA2 [Den03], *in QROM*

- Generate NTRU keypair
- Encapsulate:
  1. **Encrypt** m to randomized ciphertext
- Decapsulate:
  1. **Decrypt** to obtain m
  2. Re-**encrypt** m to verify correctness

Some XOF calls, some additional data for QROM

Operations of interest

- Sampling in $S/3$ (**K**, **E**)

Operations of interest

- Sampling in $S/3$ (**K**, **E**)

- Multiplication in R/q (**K**, **E**, **D**)
- Multiplication in $S/3$ (**D**)
- Inversion in R/q (**K**)
- Inversion in $S/3$ (**K**)

Operations of interest

- Sampling in $S/3$ (**K**, **E**)
- Multiplication in R/q (**K**, **E**, **D**)
- Multiplication in $S/3$ (**D**)
- Inversion in R/q (**K**)
- Inversion in $S/3$ (**K**)
- Lift from $S/3$ to R/q (**K**, **E**)
- Modular arithmetic (**K**, **E**, **D**)

Operations of interest

- Sampling in $S/3$ (**K**, **E**)

- Multiplication in R/q (**K**, **E**, **D**) $\triangleleft$
- Multiplication in $S/3$ (**D**)
- Inversion in R/q (**K**) $\triangleleft$
- Inversion in $S/3$ (**K**)

- Lift from $S/3$ to R/q (**K**, **E**)
- Modular arithmetic (**K**, **E**, **D**)

Operations of interest

- Sampling in $S/3$ (**K**, **E**)
- Multiplication in R/q (**K**, **E**, **D**) $\triangleleft$
- Multiplication in $S/3$ (**D**)
- Inversion in R/q (**K**) $\triangleleft$
- Inversion in $S/3$ (**K**)
- Lift from $S/3$ to R/q (**K**, **E**)
- Modular arithmetic (**K**, **E**, **D**)
- Target platform: Intel Haswell, AVX2

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication
- Get dimensions close to (multiples of) 16

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication
- Get dimensions close to (multiples of) 16

- Toom-4: 7 mults, 176 coeffs.

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication
- Get dimensions close to (multiples of) 16

- Toom-4: 7 mults, 176 coeffs.
- Karatsuba: $7 \cdot 3 = 21$ mults, 88 coeffs.

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication
- Get dimensions close to (multiples of) 16

- Toom-4: 7 mults, 176 coeffs.
- Karatsuba: $7 \cdot 3 = 21$ mults, 88 coeffs.
- Karatsuba: $21 \cdot 3 = 63$ mults, 44 coeffs.

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication
- Get dimensions close to (multiples of) 16

- Toom-4: 7 mults, 176 coeffs.
- Karatsuba: $7 \cdot 3 = 21$ mults, 88 coeffs.
- Karatsuba: $21 \cdot 3 = 63$ mults, 44 coeffs.
- Transpose. $63 \approx 64 = 4 \cdot 16$ multiplications in parallel

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication
- Get dimensions close to (multiples of) 16

- Toom-4: 7 mults, 176 coeffs.
- Karatsuba: $7 \cdot 3 = 21$ mults, 88 coeffs.
- Karatsuba: $21 \cdot 3 = 63$ mults, 44 coeffs.
- Transpose. $63 \approx 64 = 4 \cdot 16$ multiplications in parallel
- 3x Karatsuba: 22, 11 and 5/6 coeffs.
- Schoolbook multiplication fits in registers (16x parallel)

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication
- Get dimensions close to (multiples of) 16

- Toom-4: 7 mults, 176 coeffs.
- Karatsuba: $7 \cdot 3 = 21$ mults, 88 coeffs.
- Karatsuba: $21 \cdot 3 = 63$ mults, 44 coeffs.
- Transpose. $63 \approx 64 = 4 \cdot 16$ multiplications in parallel
- 3x Karatsuba: 22, 11 and 5/6 coeffs.
- Schoolbook multiplication fits in registers (16x parallel)

Optimized AVX2 assembly: 11 722 cycles

Multiplication in R/q

Goal: multiply polynomials with 701, coeffs. in $\mathbb{Z}/8192$

- 16x 16-bit words per vector register
- Toom-Cook and Karatsuba multiplication
- Get dimensions close to (multiples of) 16

- Toom-4: 7 mults, 176 coeffs.
- Karatsuba: $7 \cdot 3 = 21$ mults, 88 coeffs.
- Karatsuba: $21 \cdot 3 = 63$ mults, 44 coeffs.
- Transpose. $63 \approx 64 = 4 \cdot 16$ multiplications in parallel
- 3x Karatsuba: 22, 11 and 5/6 coeffs.
- Schoolbook multiplication fits in registers (16x parallel)

Optimized AVX2 assembly: 11 722 cycles ✓

Inversion in R/q

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

Inversion in R/q

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

- Newton iteration: invert in $R/2$, scale to $R/q = R/2^{13}$
  - At the cost of 8 multiplications in R/q [Sil99]

Inversion in R/q

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

- Newton iteration: invert in $R/2$, scale to $R/q = R/2^{13}$
  - At the cost of 8 multiplications in R/q [Sil99]

New goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/2$

Inversion in $R/2$

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/2$

Inversion in $R/2$

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Fermat's little theorem: $f^{2^{n-1}-1} \equiv 1$, so $f^{-1} \equiv f^{2^{700}-2}$

Inversion in $R/2$

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Fermat's little theorem: $f^{2^{n-1}-1} \equiv 1$, so $f^{-1} \equiv f^{2^{700}-2}$
- Itoh-Tsujii inversion
  - 12 multiplications in $R/2$
  - 13 multi-squarings (i.e. to the power 2^m) in $R/2$

Inversion in $R/2$

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Fermat's little theorem: $f^{2^{n-1}-1} \equiv 1$, so $f^{-1} \equiv f^{2^{700}-2}$
- Itoh-Tsujii inversion
  - 12 multiplications in $R/2$
  - 13 multi-squarings (i.e. to the power 2^m) in $R/2$

New goal: multiply polynomials with 701 coeffs. in $\mathbb{Z}/2$
New goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

Multiplication in $R/2$

Goal: multiply polynomials with 701 coeffs. in $\mathbb{Z}/2$

Multiplication in $R/2$

Goal: multiply polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Modern Intel CPUs: `CLMUL` instructions
  - `vpclmulqdq`: Multiply 64-coeffs. polynomials over $\mathbb{Z}/2$

Multiplication in $R/2$

Goal: multiply polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Modern Intel CPUs: `CLMUL` instructions
  - `vpclmulqdq`: Multiply 64-coeffs. polynomials over $\mathbb{Z}/2$
- Degree-3 Karatsuba: 6 mults, 234 coeffs.

Multiplication in $R/2$

Goal: multiply polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Modern Intel CPUs: `CLMUL` instructions
  - `vpclmulqdq`: Multiply 64-coeffs. polynomials over $\mathbb{Z}/2$
- Degree-3 Karatsuba: 6 mults, 234 coeffs.
- Karatsuba: $6 \cdot 3 = 18$ mults, 117 coeffs.

Multiplication in $R/2$

Goal: multiply polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Modern Intel CPUs: `CLMUL` instructions
  - `vpclmulqdq`: Multiply 64-coeffs. polynomials over $\mathbb{Z}/2$
- Degree-3 Karatsuba: 6 mults, 234 coeffs.
- Karatsuba: $6 \cdot 3 = 18$ mults, 117 coeffs.
- Schoolbook: $18 \cdot 4 = 72$ mults, $59 \approx 64$ coeffs.

Multiplication in $R/2$

Goal: multiply polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Modern Intel CPUs: `CLMUL` instructions
  - `vpclmulqdq`: Multiply 64-coeffs. polynomials over $\mathbb{Z}/2$
- Degree-3 Karatsuba: 6 mults, 234 coeffs.
- Karatsuba: $6 \cdot 3 = 18$ mults, 117 coeffs.
- Schoolbook: $18 \cdot 4 = 72$ mults, $59 \approx 64$ coeffs.

Optimized AVX2 assembly: 244 cycles

- Careful interleaving: no register spills

Multiplication in $R/2$

Goal: multiply polynomials with 701 coeffs. in $\mathbb{Z}/2$

- Modern Intel CPUs: `CLMUL` instructions
  - `vpclmulqdq`: Multiply 64-coeffs. polynomials over $\mathbb{Z}/2$
- Degree-3 Karatsuba: 6 mults, 234 coeffs.
- Karatsuba: $6 \cdot 3 = 18$ mults, 117 coeffs.
- Schoolbook: $18 \cdot 4 = 72$ mults, $59 \approx 64$ coeffs.

Optimized AVX2 assembly: 244 cycles

- Careful interleaving: no register spills

$\checkmark$

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

- It's actually about permuting bits!
- Example: binary polynomials mod $(x^7 - 1)$

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

- It's actually about permuting bits!
- Example: binary polynomials mod $(x^7 - 1)$

$f = x^6 + x^5 + x^3 + x + 1$ 0000 0000 0110 1011

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

- It's actually about permuting bits!
- Example: binary polynomials mod $(x^7 - 1)$

$f \;= x^6 + x^5 + x^3 + x + 1$ 0000 0000 0110 1011

$f^2 = x^{12} + 2x^{11} + x^{10} + 2x^9 + 2x^8 + 2x^7 + 5x^6 + 2x^5 + 2x^4 + 2x^3 + x^2 + 2x + 1$

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

- It's actually about permuting bits!
- Example: binary polynomials mod $(x^7 - 1)$

$$f = x^6 + x^5 + x^3 + x + 1 \qquad 0000\,0000\,0110\,1011$$

$$f^2 = x^{12} + 2x^{11} + x^{10} + 2x^9 + 2x^8 + 2x^7 + 5x^6 + 2x^5 + 2x^4 + 2x^3 + x^2 + 2x + 1$$

$$\equiv x^{12} + x^{10} + x^6 + x^2 + 1 \qquad 0001\,0100\,0100\,0101$$

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

- It's actually about permuting bits!
- Example: binary polynomials mod $(x^7 - 1)$

$$f = x^6 + x^5 + x^3 + x + 1 \qquad 0000\,0000\,0110\,1011$$

$$f^2 = x^{12} + 2x^{11} + x^{10} + 2x^9 + 2x^8 + 2x^7 + 5x^6 + 2x^5 + 2x^4 + 2x^3 + x^2 + 2x + 1$$

$$\equiv x^{12} + x^{10} + x^6 + x^2 + 1 \qquad 0001\,0100\,0100\,0101$$

$$\ldots \to \ 0\,0010\,1000$$

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

- It's actually about permuting bits!
- Example: binary polynomials mod $(x^7 - 1)$

$$
\begin{aligned}
f &= x^6 + x^5 + x^3 + x + 1 && 0000\,0000\,0110\,1011 \\
f^2 &= x^{12} + 2x^{11} + x^{10} + 2x^9 + 2x^8 + 2x^7 + 5x^6 + 2x^5 + 2x^4 + 2x^3 + x^2 + 2x + 1 && \\
&\equiv x^{12} + x^{10} + x^6 + x^2 + 1 && 0001\,0100\,0100\,0101 \\
& && \ldots \rightarrow\ 0\,0010\,1000 \\
&\equiv x^6 + x^5 + x^3 + x^2 + 1 && 0000\,0000\,0110\,1101
\end{aligned}
$$

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

- It's actually about permuting bits!
- Example: binary polynomials mod $(x^7 - 1)$

$$f = x^6 + x^5 + x^3 + x + 1 \qquad 0000\,0000\,0110\,1011$$

$$f^2 = x^{12} + 2x^{11} + x^{10} + 2x^9 + 2x^8 + 2x^7 + 5x^6 + 2x^5 + 2x^4 + 2x^3 + x^2 + 2x + 1$$

$$\equiv x^{12} + x^{10} + x^6 + x^2 + 1 \qquad 0001\,0100\,0100\,0101$$

$$\ldots \rightarrow \; 0\,0010\,1000$$

$$\equiv x^6 + x^5 + x^3 + x^2 + 1 \qquad 0000\,0000\,0110\,1101$$

- Observation: multi-squarings are composed permutations

Multi-squaring in $R/2$

Goal: (multi-)square polynomials with 701 coeffs. in $\mathbb{Z}/2$

- It's actually about permuting bits!
- Example: binary polynomials mod $(x^7 - 1)$

$$
\begin{aligned}
f &= x^6 + x^5 + x^3 + x + 1 && \texttt{0000 0000 0110 1011} \\
f^2 &= x^{12} + 2x^{11} + x^{10} + 2x^9 + 2x^8 + 2x^7 + 5x^6 + 2x^5 + 2x^4 + 2x^3 + x^2 + 2x + 1 && \\
&\equiv x^{12} + x^{10} + x^6 + x^2 + 1 && \texttt{0001 0100 0100 0101} \\
& && \ldots \rightarrow \texttt{0 0010 1000} \\
&\equiv x^6 + x^5 + x^3 + x^2 + 1 && \texttt{0000 0000 0110 1101}
\end{aligned}
$$

- Observation: multi-squarings are composed permutations
  - $\Rightarrow$ Still 'just' permutations

New Goal: permutations on 701 bits

Permuting bits with AVX2

- Dedicated routines.. or generated assembly

Permuting bits with AVX2

- Dedicated routines.. or generated assembly
- Python tool: simulate relevant subset of AVX2
  - Show bits by index, not by value
  - Interactively create permutations, or generate

Permuting bits with AVX2

- Dedicated routines.. or generated assembly
- Python tool: simulate relevant subset of AVX2
  - Show bits by index, not by value
  - Interactively create permutations, or generate

1. Using `pext` and `pdep` (`BMI2` instructions)
   - Based on patience-sort
   - Relabel, find longest increasing sequences
   - More efficient for structured permutations

Permuting bits with AVX2

- Dedicated routines.. or generated assembly

- Python tool: simulate relevant subset of AVX2
  - Show bits by index, not by value
  - Interactively create permutations, or generate

1. Using `pext` and `pdep` (`BMI2` instructions)
   - Based on patience-sort
   - Relabel, find longest increasing sequences
   - More efficient for structured permutations
2. Using `vpshufb` and `vpermq`
   - Bytewise shuffling, masking
   - Fairly uniform performance

Permuting bits with AVX2

- Dedicated routines.. or generated assembly
- Python tool: simulate relevant subset of AVX2
  - Show bits by index, not by value
  - Interactively create permutations, or generate

1. Using `pext` and `pdep` (`BMI2` instructions)
   - Based on patience-sort
   - Relabel, find longest increasing sequences
   - More efficient for structured permutations
2. Using `vpshufb` and `vpermq`
   - Bytewise shuffling, masking
   - Fairly uniform performance

Single squaring:	58 cycles
Average multi-squaring:	235 cycles

Permuting bits with AVX2

- Dedicated routines.. or generated assembly

- Python tool: simulate relevant subset of AVX2
  - Show bits by index, not by value
  - Interactively create permutations, or generate

1. Using `pext` and `pdep` (`BMI2` instructions)
   - Based on patience-sort
   - Relabel, find longest increasing sequences
   - More efficient for structured permutations
2. Using `vpshufb` and `vpermq`
   - Bytewise shuffling, masking
   - Fairly uniform performance

Single squaring:	58 cycles
Average multi-squaring:	235 cycles

✓

Inversion in R/q (cont.)

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

Inversion in R/q (cont.)

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

$=$ 8x mult. in R/q $+$ inversion in $R/2$

Inversion in R/q (cont.)

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

$=$ 8x mult. in R/q $+$ inversion in $R/2$
$=$ 8x mult. in R/q $+$ 12x mult. in $R/2$ $+$ 13x m.-squaring in $R/2$

Inversion in R/q (cont.)

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

$= 8$x mult. in R/q + inversion in $R/2$
$= 8$x mult. in R/q + 12x mult. in $R/2$ + 13x m.-squaring in $R/2$
$= 8$x mult. in R/q + 12x mult. in $R/2$ + 13x bit permutations

Inversion in R/q (cont.)

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

$= 8$x mult. in R/q + inversion in $R/2$
$= 8$x mult. in R/q + 12x mult. in $R/2$ + 13x m.-squaring in $R/2$
$= 8$x mult. in R/q + 12x mult. in $R/2$ + 13x bit permutations

Multiplication in R/q:	11 722 cycles
Inversion in $R/2$:	10 322 cycles

Inversion in R/q (cont.)

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

$= 8\text{x mult. in } R/q + \text{inversion in } R/2$
$= 8\text{x mult. in } R/q + 12\text{x mult. in } R/2 + 13\text{x m.-squaring in } R/2$
$= 8\text{x mult. in } R/q + 12\text{x mult. in } R/2 + 13\text{x bit permutations}$

Multiplication in R/q: 11 722 cycles
Inversion in $R/2$: 10 322 cycles

Inversion in R/q: 107 726 cycles

- Includes some cost for conversions

Inversion in R/q (cont.)

Goal: invert polynomials with 701 coeffs. in $\mathbb{Z}/8192$

$= 8$x mult. in R/q + inversion in $R/2$
$= 8$x mult. in R/q + 12x mult. in $R/2$ + 13x m.-squaring in $R/2$
$= 8$x mult. in R/q + 12x mult. in $R/2$ + 13x bit permutations

Multiplication in R/q:	11 722 cycles
Inversion in $R/2$:	10 322 cycles

Inversion in R/q: 107 726 cycles

- Includes some cost for conversions

✓

Results

- Encapsulation: <u>48 646</u> cycles
  - R/q multiplication (11 722)
  - sampling, conversions, SHAKE128

Results

- Encapsulation: 48 646 cycles
  - R/q multiplication (11 722)
  - sampling, conversions, SHAKE128

- Decapsulation: 67 338 cycles
  - $S/3$ & R/q multiplication (2x 11 722)
  - encrypt (R/q multiplication, sampling)
  - conversions, SHAKE128

Results

- Encapsulation: 48 646 cycles
  - R/q multiplication (11 722)
  - sampling, conversions, SHAKE128
- Decapsulation: 67 338 cycles
  - $S/3$ & R/q multiplication (2x 11 722)
  - encrypt (R/q multiplication, sampling)
  - conversions, SHAKE128
- Key generation: 307 914 cycles
  - $S/3$ inversion (159 606)
  - R/q inversion (107 726)
  - R/q multiplication (11 722)
  - sampling, conversions

Results

- Encapsulation: 48 646 cycles
  - R/q multiplication (11 722)
  - sampling, conversions, SHAKE128
- Decapsulation: 67 338 cycles
  - $S/3$ & R/q multiplication (2x 11 722)
  - encrypt (R/q multiplication, sampling)
  - conversions, SHAKE128
- Key generation: 307 914 cycles
  - $S/3$ inversion (159 606)
  - R/q inversion (107 726)
  - R/q multiplication (11 722)
  - sampling, conversions
- Benchmarks on Intel Core i7-4770K (Haswell) at 3.5GHz
  - Keygen: ~0.1ms, Encaps/Decaps: ~0.02ms

Comparison

- Comparison is hard: assumptions and optimizations vary
  - See paper for footnotes

	K	E	D	pk	sk	ct
			Passively secure KEMs			
BCNS	2.5*m*	4.0*m*	482*k*	4096	4096	4224
NewHope	89*k*	111*k*	19*k*	1792	1824	2048
Frodo	2.9*m*	3.5*m*	338*k*	11.3*k*	11.3*k*	11.3k
			CCA2-secure KEMs			
Streamlined NTRU Prime 4591^{761}	6.1*m*	60*k*	97*k*	1600	1218	1047
spLWE-KEM	337*k*	814*k*	785*k*	?	?	804
Kyber	78*k*	120*k*	126*k*	2400	1088	1184
NTRU-KEM (this work)	**308k**	**49k**	**67k**	**1422**	**1140**	**1281**
			CCA2-secure public-key encryption			
NTRU ees743ep1	1.2*m*	57*k*	111*k*	1120	1027	980
Lizard	98*m*	35*k*	81*k*	467*k*	2.0*m*	1072

Takeaway

- When choosing the right parameters ..
- .. constant time key generation can be fast
  - .. not just encryption / decryption;
- .. and constant time sampling can be fast
- .. without decryption failures

- NTRU can be a fast ephemeral CCA2-secure KEM

Takeaway

- When choosing the right parameters ..
- .. constant time key generation can be fast
  - .. not just encryption / decryption;
- .. and constant time sampling can be fast
- .. without decryption failures

- NTRU can be a fast ephemeral CCA2-secure KEM

- Code is available (CC0 Public Domain): `https://joostrijneveld.nl/papers/ntrukem`
- Bit permutations tool included (CC0 Public Domain): `https://joostrijneveld.nl/code/bitpermutations`

References I

Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe.
Post-quantum key exchange – a new hope.
In Thorsten Holz and Stefan Savage, editors, *Proceedings of the 25th USENIX Security Symposium*. USENIX Association, 2016.
https://cryptojedi.org/papers/#newhope.

Joppe Bos, Craig Costello, Leo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila.
Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE.
In Christopher Kruegel, Andrew Myers, and Shai Halevi, editors, *Conference on Computer and Communications Security – CCS '16*, pages 1006–1018. ACM, 2016.
https://doi.org/10.1145/2976749.2978425.

References II

Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal.
NTRU Prime.
In Jan Camenisch and Carlisle Adams, editors, *Selected Areas in Cryptography – SAC 2017*, LNCS, to appear. Springer, 2017.
http://ntruprime.cr.yp.to/papers.html.

Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila.
Post-quantum key exchange for the TLS protocol from the ring learning with errors problem.
In Lujo Bauer and Vitaly Shmatikov, editors, *2015 IEEE Symposium on Security and Privacy*, pages 553–570. IEEE, 2015.
https://eprint.iacr.org/2014/599.

Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé.
CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM.
Cryptology ePrint Archive, Report 2017/634, 2017.
http://eprint.iacr.org/2017/634.

References III

Jung Hee Cheon, Kyoohyung Han, Jinsu Kim, Changmin Lee, and Yongha Son.

A practical post-quantum public-key cryptosystem based on spLWE.

In Seokhie Hong and Jong Hwan Park, editors, *Information Security and Cryptology – ICISC 2016*, volume 10157 of *LNCS*, pages 51–74. Springer, 2017.

https://eprint.iacr.org/2016/1055.

Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song.

Lizard: Cut off the tail! Practical post-quantum public-key encryption from LWE and LWR.

IACR Cryptology ePrint Archive report 2016/1126, 2016.

https://eprint.iacr.org/2016/1126.

Alexander W. Dent.

A designer's guide to KEMs.

In Kenneth G. Paterson, editor, *Cryptography and Coding*, volume 2898 of *LNCS*, pages 133–151. Springer, 2003.

http://www.cogentcryptography.com/papers/designer.pdf.

References IV

Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman.
NTRU: A ring-based public key cryptosystem.
In Joe P. Buhler, editor, *Algorithmic Number Theory – ANTS-III*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998.
http://dx.doi.org/10.1007/BFb0054868.

Joseph H. Silverman.
Almost inverses and fast NTRU key creation.
Technical Report #014, NTRU Cryptosystems, 1999.
Version 1. https://assets.onboardsecurity.com/static/downloads/NTRU/resources/NTRUTech014.pdf.

Multiplication in $S/3$

Goal: multiply polynomials with 700 coeffs. in $\mathbb{Z}/3$

Multiplication in $S/3$

Goal: multiply polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Bitslice 2-bit coeffients
- Get dimensions close to (multiples of) 256

Multiplication in $S/3$

Goal: multiply polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Bitslice 2-bit coeffients
- Get dimensions close to (multiples of) 256

- 5x Karatsuba, 253 mults of 22 coeffs.?
- Then 256x parallel schoolbook? Or more Karatsuba?

Multiplication in $S/3$

Goal: multiply polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Bitslice 2-bit coeffients
- Get dimensions close to (multiples of) 256

- 5x Karatsuba, 253 mults of 22 coeffs.?
- Then 256x parallel schoolbook? Or more Karatsuba?

- Re-use multiplication in R/q
- Each product term stays well below $q = 8192$

Multiplication in $S/3$

Goal: multiply polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Bitslice 2-bit coeffients
- Get dimensions close to (multiples of) 256

- 5x Karatsuba, 253 mults of 22 coeffs.?
- Then 256x parallel schoolbook? Or more Karatsuba?

- Re-use multiplication in R/q
- Each product term stays well below $q = 8192$
- Not optimal, but close enough and easier

Multiplication in $S/3$

Goal: multiply polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Bitslice 2-bit coeffients
- Get dimensions close to (multiples of) 256

- 5x Karatsuba, 253 mults of 22 coeffs.?
- Then 256x parallel schoolbook? Or more Karatsuba?

- Re-use multiplication in R/q
- Each product term stays well below $q = 8192$
- Not optimal, but close enough and easier

✓

Inversion in $S/3$

Goal: invert polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Use 'almost inverse' algorithm [Sil99]
  - Can be seen as EGCD for $S/3$
  - Inherently not constant time
  - Ref. C code: also use this for $R/2$

Inversion in $S/3$

Goal: invert polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Use 'almost inverse' algorithm [Sil99]
  - Can be seen as EGCD for $S/3$
  - Inherently not constant time
  - Ref. C code: also use this for $R/2$
- Make constant time!

Inversion in $S/3$

Goal: invert polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Use 'almost inverse' algorithm [Sil99]
  - Can be seen as EGCD for $S/3$
  - Inherently not constant time
  - Ref. C code: also use this for $R/2$
- Make constant time!
- Divide by x, multiply, add — for every coefficient
  - 1400 iterations (as opposed to average ~933)
  - Always swap f and g

Inversion in $S/3$

Goal: invert polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Use 'almost inverse' algorithm [Sil99]
  - Can be seen as EGCD for $S/3$
  - Inherently not constant time
  - Ref. C code: also use this for $R/2$
- Make constant time!
- Divide by x, multiply, add — for every coefficient
  - 1400 iterations (as opposed to average ~933)
  - Always swap f and g
- Truncated, bit-sliced vectors of coefficients

Inversion in $S/3$

Goal: invert polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Use 'almost inverse' algorithm [Sil99]
  - Can be seen as EGCD for $S/3$
  - Inherently not constant time
  - Ref. C code: also use this for $R/2$
- Make constant time!
- Divide by x, multiply, add — for every coefficient
  - 1400 iterations (as opposed to average ~933)
  - Always swap f and g
- Truncated, bit-sliced vectors of coefficients

Inversion in $S/3$: 159 606 cycles

Inversion in $S/3$

Goal: invert polynomials with 700 coeffs. in $\mathbb{Z}/3$

- Use 'almost inverse' algorithm [Sil99]
  - Can be seen as EGCD for $S/3$
  - Inherently not constant time
  - Ref. C code: also use this for $R/2$
- Make constant time!
- Divide by x, multiply, add — for every coefficient
  - 1400 iterations (as opposed to average ~933)
  - Always swap f and g
- Truncated, bit-sliced vectors of coefficients

Inversion in $S/3$: 159 606 cycles

✓

Encapsulate and decapsulate

Encaps (h)

1: $c_0 \leftarrow \{0,1\}^{\mu}$
2: $m = \mathsf{Sample}\mathcal{T}(c_0)$
3: $c_1 = \mathsf{XOF}(m, \mu, \mathtt{coins})$
4: $k = \mathsf{XOF}(m, \mu, \mathtt{key})$
5: $e_1 = \mathcal{E}(m, c_1, h)$
6: $e_2 = \mathsf{XOF}(m, \mathit{len}(m), \mathtt{qrom})$

Output: Ciphertext (e_1, e_2), session key k.

Decaps $((e_1, e_2), (f, h))$

1: $m = \mathcal{D}(e, f)$
2: $c_1 = \mathsf{XOF}(m, \mu, \mathtt{coins})$
3: $k = \mathsf{XOF}(m, \mu, \mathtt{key})$
4: $e_1' = \mathcal{E}(m, c_1, h)$
5: $e_2' = \mathsf{XOF}(m, \mathit{len}(m), \mathtt{qrom})$
6: **if** $(e_1', e_2') \neq (e_1, e_2)$ **then**
7: $\quad k = \perp$
8: **end if**

Output: Session key k