Shorter Messages and Faster Post-Quantum Encryption with Round5 on Cortex M

Markku-Juhani O. Saarinen <mjos@pqshield.com>

S. Bhattacharya[1] O. Garcia-Morchon[1] R. Rietman[1] L. Tolhuizen[1] Z. Zhang[2]

(1) Philips Research, NL (2) Algorand, USA (Prev. OnBoard Security)

CARDIS 2018

13 November 2018 – Montpellier, France

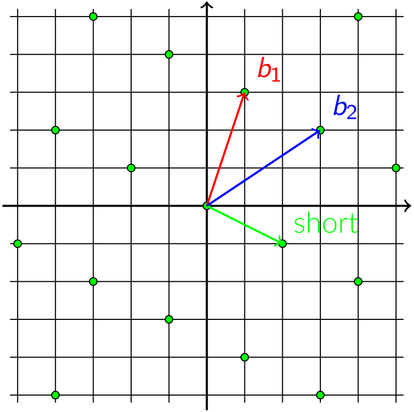
Round2 + Hila5 = Round5

Dustin Moody (NIST) on April 30, 2018:

> *"NIST would like to encourage any submissions which are quite similar to consider merging. It would be helpful if any such merger is announced (to NIST) before November 30th. [..] While the selection of candidates for the second round will primarily be based on the original submissions, NIST may consider a merged submission more attractive than either of the original schemes if it provides improvements in security, efficiency, or compactness and generality of presentation."*

- We took the NIST advice and decided to merge my personal round 1 candidate **Hila5** with **Round2**, primarily from Philips. The resulting **Round5** algorithm has better security analysis and is much more efficient that either one of its parents.
- Round5 resembles the ring variants of Round2 more than Hila5. Hila5 was based on Ring-LWE in anticyclic $x^n + 1$ "NewHope" ring ($n = 1024$, $q = 12289$), but with a novel reconciliation method - and importantly **XEf error correction codes**.

General and Ideal Lattices



Lattice $\mathbb{Z}^2$ with basis (b_1, b_2).

Lattice basis:

$$\mathcal{L}(\mathrm{b}_1, \ldots, \mathrm{b}_\mathrm{n}) = \sum_{i=1}^{n} x_i \mathrm{b}_\mathrm{i} \mid x_i \in \mathbb{Z}$$

If we replace b_2 with the shorter green vector, same lattice is generated. This is a **reduced basis**.

Shortest Vector Problem (**SVP**) is the task of finding the shortest vector in a lattice. Related hard lattice problems include: Bounded Distance Decoding (**BDD**), Short Integer Solution (**SIS**), Learning With Errors (**LWE**), and **Learning With Rounding** (**LWR**).

In order to go from n^2 variables to n variables, one may use **ideal lattices**, which are defined in **rings**.

(G)LWR vs (R/M)LWE

Hila5, Kyber, NewHope, Frodo, etc. are based on versions of the *Learning With Errors* (**LWE**) problem, which requires addition of "noise" sampled from a bell-shaped (e.g. Gaussian or Binomial) distribution to create hard instances.

(G)LWR: Noise from Rounding (no RNG)

Round5, Round2, Saber instead use versions of *Learning With Rounding* (**LWR**).
No sampling is required in LWR; "noise" comes directly from rounding in the form of a lossy compression function we may call **Round**. It maps $x \in \mathbb{Z}_a$ to $\mathbb{Z}_b$ using h:

$$\mathrm{Round}_{a \to b}(x, h) = \left\lfloor \frac{b}{a} \cdot x + h \right\rfloor \bmod b. \tag{1}$$

This is equivalent to rounding of $^b/_a \cdot x$ to closest integer when rounding constant $h = {}^1/_2$. Each coefficient is operated on separately when **Round** or modular reduction ("**mod**") is applied to polynomials, vectors, or matrices.

Hila5: Ring-LWE (Learning With Errors – in a Ring)

Hila5 used the Ring Learning With Errors (RLWE) problem. Let $\mathcal{R}$ be a ring with elements $\mathbf{v} \in \mathbb{Z}_q^n$. As usual, $n = 2^m$ power of two and $n \mid q - 1$ for **NTT** to work.

Definition (RLWE – Informal)

With all distributions and computations in ring $\mathcal{R}$, let $\mathbf{s}, \mathbf{e}$ be elements randomly chosen from some non-uniform distribution χ, and $\mathbf{g}$ be a uniformly random public value. Determining $\mathbf{s}$ from $(\mathbf{g}, \mathbf{g} * \mathbf{s} + \mathbf{e})$ in ring $\mathcal{R}$ is the (Normal Form Search) Ring Learning With Errors (RLWE$_{\mathcal{R},\chi}$) problem.

The hardness of the problem is a function of n, q, and χ. Typically χ is chosen so that each coefficient is a Gaussian or from some other "Bell-Shaped" distribution that is relatively tightly concentrated around zero. Hila5 used a binomial "bitcount" χ.

Round2, Round5: General Learning With Rounding

Definition (General LWR (GLWR) – Informal)

Let d, n, p, q be positive integers such that $q \geq p \geq 2$, and $n \in \{ 1, d \}$. Let $\mathcal{R}_{n,q}$ be a polynomial ring, and let D_s be a probability distribution on $\mathcal{R}_n^{d/n}$.

- The search version of the GLWR problem $\mathrm{sGLWR}_{d,n,m,q,p}(D_s)$ is as follows: given m samples of the form $(\mathrm{a}_i, b_i = \mathrm{Round}_{q \to p}(\mathrm{a}_i^T \mathrm{s} \bmod q, 1/2)$ with $\mathrm{a}_i \in \mathcal{R}_{n,q}^{d/n}$ and a fixed $\mathrm{s} \leftarrow D_s$, recover s.
- The decision GLWR problem $\mathrm{dGLWR}_{d,n,m,q,p}(D_s)$ is to distinguish between the uniform distribution on $\mathcal{R}_{n,q}^{d/n} \times \mathcal{R}_{n,p}$ and the distribution $(\mathrm{a}_i, b_i = \mathrm{Round}_{q \to p}(\mathrm{a}_i^T \mathrm{s}_i \bmod q, 1/2))$ with $\mathrm{a}_i \leftarrow \mathcal{R}_{n,q}^{d/n}$ and a fixed $s \leftarrow D_s$.

Here $n = d$ implies the ring variant that we are using in Round5. Typically p, q are powers of two (no modular reduction) and there is more freedom in choosing n.

Round5 rough high-level description: Based on "Noisy ElGamal"

- Round5 comes in **both** ring and non-ring variants – in this work we will focus on faster and smaller ring ($n = d$) variants on low-end microcontrollers (Cortex M).
- **XEf** is a forward error correction code that can be easily implemented in constant time fashion (this comes from Hila5).
- **Two rings** are used: $x^{n+1} - 1$, with $n + 1$ prime (cyclic, a bit like HILA5), and its subring $\Phi_{n+1} = (x^{n+1} - 1)/(x - 1) = x^n + \cdots + x + 1$ (resembling Round2).
- For small-norm secrets we use "balanced" sparse ternary polynomials $D \subset \{-1, 0, 1\}^n$, with $h/2$ of both $+1$ and -1 coefficients ($n - h$ set to 0).
- The CPA versions `R5ND_nKEM` are intended for key establishment while CCA versions `R5ND_nPKE` are for public key encryption. $n \in \{1, 3, 5\}$ is the NIST security level (corresponding to security of AES with $\{128, 192, 256\}$ bit key).
- The CCA KEM can be used as Public Key Encryption Algorithm with a DEM: SHAKE-256 to expands the shared secret to AES-GCM key and IV for data.

Round5 Ring Variants: Simple Key Generation

<u>KeyGenCPA(σ, γ): Key generation for CPA case.</u>

Require: Random seeds σ, γ.

1: $\mathsf{a} \xleftarrow{\$_\sigma} \mathbb{Z}_q^n$	*Uniform polynomial, seed* σ.
2: $\mathsf{s} \xleftarrow{\$_\gamma} D$	*Sparse ternary polynomial, seed* γ.
3: $\mathsf{b} \leftarrow \mathsf{Round}_{q \to p}(\mathsf{a} * \mathsf{s} \mod \Phi_{n+1}, 1/2)$	*Compress product to range* $0 \leq b_i < p$.
4: $\mathtt{sk} = \mathsf{s}$	s: *Random seed* γ *is sufficient.*
5: $\mathtt{pk} = (\mathsf{a}, \mathsf{b})$	a: *Random seed* σ, b: $n \log_2 p$ *bits.*

Ensure: Public key $\mathtt{pk} = (\mathsf{a}, \mathsf{b})$ and secret key $\mathtt{sk} = \mathsf{s}$.

<u>Note</u>: This description is simplified; e.g. the rounding constants are not always $1/2$.

Round5 Ring Variants: Encryption uses (XEf) Error Correction

EncryptCPA($\mathsf{m}, \mathtt{pk}, \rho$): Public key encryption (CPA).

Require: Message $\mathsf{m} = \{0,1\}^m$, public key $\mathtt{pk} = (\mathsf{a}, \mathsf{b})$, random seed ρ.

1: $\mathsf{r} \xleftarrow{\$_\rho} D$ — *Sparse ternary polynomial, seed* ρ

2: $\mathsf{u} \leftarrow \mathsf{Round}_{q \to p}(\mathsf{a} * \mathsf{r} \mod \Phi_{n+1}, 1/2)$ — *Compress product to range* $0 \le u_i < p$.

3: $\mathsf{t} \leftarrow \mathsf{Sample}_\mu(\mathsf{b} * \mathsf{r} \mod x^{n+1} - 1)$ — *Noisy shared secret, truncate to* $\mathbb{Z}_p^\mu$.

4: $\mathsf{v} \leftarrow \mathsf{Round}_{p \to t}(\mathsf{t} + \frac{p}{2}\mathsf{m}, 1/2)$ — *Add message + XEf* $\mathsf{m} \in \mathbb{Z}_2^\mu$.

5: $\mathtt{ct} = (\mathsf{u}, \mathsf{v})$ — u: $n \log_2 p$ *bits*, v: $\mu \log_2 t$ *bits*.

Ensure: Ciphertext $\mathtt{ct} = (\mathsf{u}, \mathsf{v})$.

The CPA scheme is transformed into a chosen-ciphertext (IND-CCA2) secure one (`R5ND_xPKEb`) using the usual Fujisaki-Okamoto Transform [FuOk99,HoHoKi17].

Decryption

<u>DecryptCPA($\mathtt{ct}, \mathtt{sk}$): Decryption (CPA)</u>

Require: Ciphertext $\mathtt{ct} = (\mathsf{u}, \mathsf{v})$, secret key $\mathtt{sk} = \mathsf{s}$.

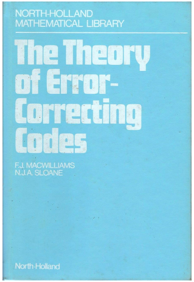
1: $\mathsf{t}' \leftarrow \mathsf{Sample}_\mu(\mathsf{u} * \mathsf{s} \mod x^{n+1} - 1)$ *Noisy shared secret, truncate to $\mathbb{Z}_p^\mu$.*

2: $\mathsf{m} \leftarrow \mathsf{Round}_{p \to 2}(\frac{p}{t}\mathsf{v} - \mathsf{t}', {}^1\!/_2)$ *Remove noise, apply XEf correction.*

Ensure: Plaintext $\mathtt{pt} = \mathsf{m}$.

- To see why the algorithm works, note that the shared secrets **EncryptCPA()** and **DecryptCPA()** satisfy approximately $\mathsf{t} \approx \mathsf{t}' \approx \mathsf{a} * \mathsf{s} * \mathsf{r}$.
- The Φ_{n+1} ring a subring of the $x^{n+1} - 1 = (x - 1) * \Phi_{n+1}$ ring. Since s and r are "balanced", their coefficients sum to zero and they are divisible by $(x - 1)$.
- High bits of t are used as a "one time pad" to transport the message payload.

Error Correction Code XEf



$\leftarrow$ **Hey kids! Pay attention in coding theory classes. It's useful.**

I chose a simple forward error correction code, XE5, for HILA5. Error correction helps to significantly shrink message sizes.

Additional Requirement: Fast, constant-time implementable.

XE5 has been generalized as **XEf** for Round5, where $f \in \{1, 2, 3, 4, 5\}$ is the number of bit flips it can always correct.

- **XEf** is a simple parity code. Both encoding and decoding are parallelizable and easy to implement efficiently on 8 to 64-bit architectures and in hardware.
- **Constant-time.** No table lookups or branches. An "error oracle" would lead to attacks under CCA security model. The KEMs also never fail (QROM proofs).

I first described similar constant-time error correction techniques (for TRUNC8) in: `https://eprint.iacr.org/2016/1058` (Original uploaded November 15, 2016)

Round5 Ring Variants for Key Establishment

	Parameters	CPA NIST1	CPA NIST3	CPA NIST5
Round5.KEM	d, n, h	490, 490, 162	756, 756, 242	940, 940, 414
	q, p, t	$2^{10}, 2^{7}, 2^{4}$	$2^{12}, 2^{8}, 2^{2}$	$2^{12}, 2^{8}, 2^{3}$
	$B, \bar{n}, \bar{m}, f$	1, 1, 1, 3	1, 1, 1, 3	1, 1, 1, 3
	μ	$128 + 91$	$192 + 103$	$256 + 121$
	Public key	445 B	780 B	972 B
	Ciphertext	539 B	830 B	1082 B
	PQ Security	2^{118}	2^{176}	2^{232}
	Classical	2^{128}	2^{193}	2^{256}
	Failure rate	2^{-78}	2^{-78}	2^{-95}
	Version ID	`R5ND_1KEMb`	`R5ND_3KEMb`	`R5ND_5KEMb`

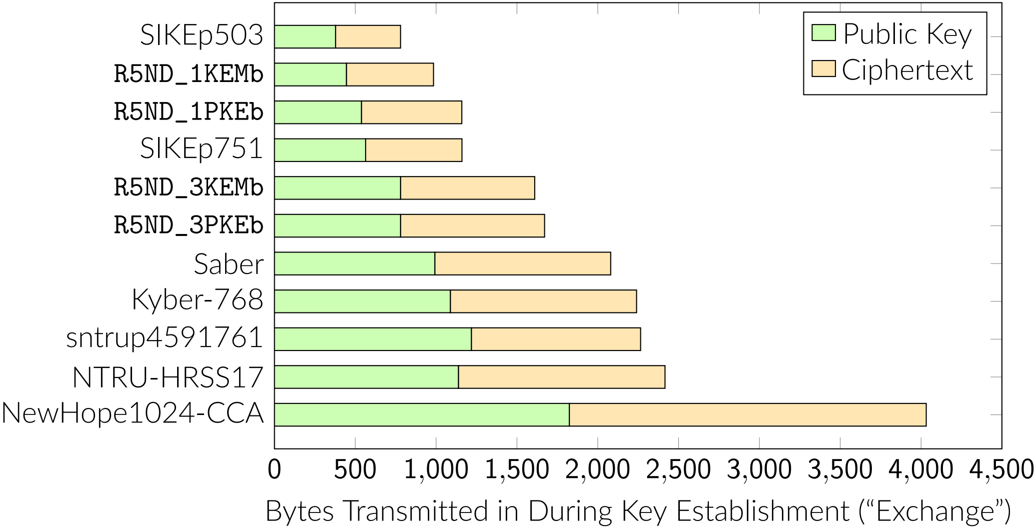
In TLS, SSH, ... Key Establishment, ephemeral keys are used, so CPA is fine. Also a higher failure probability is acceptable since the process can be simply repeated.

Round5 Ring Variants for Public Key Encryption

	Parameters	CCA NIST1	CCA NIST3	CCA NIST5
Round5.PKE	d, n, h	522, 522, 208	756, 756, 242	940, 940, 406
	q, p, t	$2^{13}, 2^{8}, 2^{3}$	$2^{12}, 2^{8}, 2^{3}$	$2^{12}, 2^{8}, 2^{4}$
	$B, \bar{n}, \bar{m}, f$	1, 1, 1, 3	1, 1, 1, 3	1, 1, 1, 3
	μ	$128 + 91$	$192 + 103$	$256 + 121$
	Public key	538 B	780 B	972 B
	Ciphertext	621 B	891 B	1161 B
	PQ Security	2^{117}	2^{176}	2^{232}
	Classical	2^{128}	2^{193}	2^{256}
	Failure rate	2^{-202}	2^{-171}	2^{-131}
	Version ID	`R5ND_1PKEb`	`R5ND_3PKEb`	`R5ND_5PKEb`

Public Key Encryption variants have IND-CCA2 security (long-term keys) and have a negligible failure probability. You can use these e.g. for e-mail or cloud backups.

Parameters are Optimized for Bandwidth



Polynomial Multiplication with Sparse Ternary Secrets

- Polynomial multiplication (and reduction) with our sparse ternary secrets of weight h degree n is requires $\Theta(nh)$ word additions / subtractions.
- Even though h is a lot smaller than n, and no word multiplications are needed, this is essentially an $O(n^2)$ algorithm.
- Since p, q, t are powers of two, no modular reduction is needed. So the complexity is this simple multiplication algorithm is $\Theta(nh \log_2 q)$ bit operations.
- Also **no word multiplications** are required; this is useful on extremely low-end CPUs such as 8-bit AVR that does not have a word $\times$ word multiplier.
- Parallel byte or word additions / subtractions resemble string operations. They are also directly supported by **SIMD** architectures (AVX2, AVX-512, ARM NEON) which makes them fast on high-end CPUs (and in **hardware**).
- In other words, the special form allows smaller implementation footprint in both hardware and software. Flexible, fast implementations are **easy to write**.

Polynomial Multiplication for General, Non-Ternary Secrets

This is much, much harder [1]:

> *"[..] systematically exploring different combinations of Toom-3, Toom-4, and Karatsuba decomposition of multiplication in $\mathcal{R}_q$, and by carefully hand-optimizing multiplication of low-degree polynomial multiplication at the bottom of the Toom/Karatsuba decomposition."*

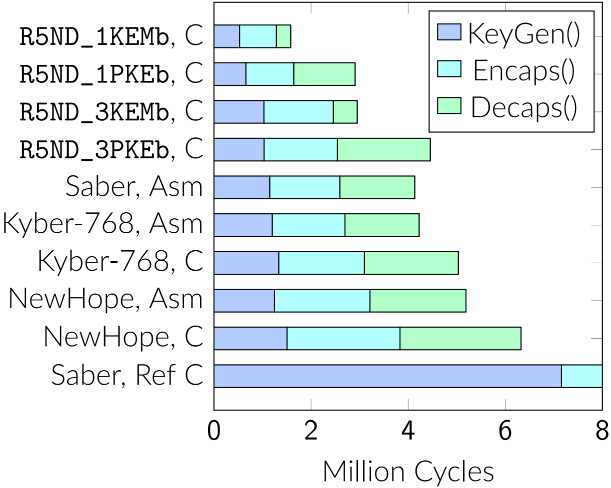
[1] M. J. Kannwischer, J. Rijneveld and P. Schwabe: "*Faster multiplication in $\mathbb{Z}_{2^m}[x]$ on Cortex-M4 to speed up NIST PQC candidates*", `https://eprint.iacr.org/2018/1018`, Oct 2018.

- NTT: $\Theta(n \log n)$ – requires smooth n and $n \mid q - 1$.
- Toom-3: $\Theta(n^{\frac{\log 5}{\log 3}}) \approx \Theta(n^{1.46497})$
- Karatsuba: $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.58496})$, etc.

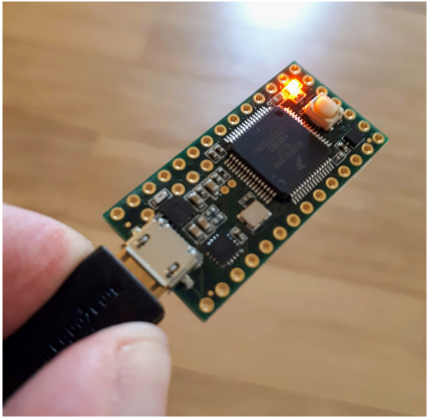
$\Theta(1)$	$=$	Θ(😎)
$\Theta(\log(n))$	$=$	Θ(😁)
$\Theta((\log(n))^c)$	$=$	Θ(😇)
$\Theta(n)$	$=$	Θ(😍)
$\Theta(n \log(n))$	$=$	Θ(🙂)
$\Theta(n^{1.5})$	$=$	Θ(😐)
$\Theta(n^2)$	$=$	Θ(☹️)
$\Theta(n^c)$	$=$	Θ(😠)
$\Theta(c^n)$	$=$	Θ(😫)
$\Theta(n!)$	$=$	Θ(😱)

The asymptotically best algorithm is not necessarily the fastest for given n.

Performance of our C implementation on Cortex M4



"NewHope" refers to NewHope1024CCA. Saber is [KaMeRo+18].



My \$20 Teensy 3.2 dev board has a (\$5) **MK20DX256** @ 96 MHz.
"Hi-spec": 64kB RAM, 256kB Flash.
(A lo-spec Cortex M4 MCU costs \$1.)

Performing Key Establishment at NIST 3 Security Level

Xfer: Public key + Ciphertext. **Time**: KeyGen + Encaps + Decaps on M4 @ 24 MHz.
Code: Size of implementation in bytes. **Fail**: Decryption failure bound.
PQ: Claimed quantum security. **Classic**: Claimed classical security.

Algorithm	Xfer	Time	Code	Fail	PQ	Classic
`R5ND_3KEMb` (C)	1610	0.123s	4464	2^{-78}	2^{176}	2^{193}
`R5ND_3PKEb` (C)	1671	0.185s	5232	2^{-171}	2^{176}	2^{193}
Saber (CHES18 Asm)	2080	0.172s	?	2^{-136}	2^{180}	2^{198}
Kyber-768 (Asm)	2240	0.210s	7016	2^{-142}	2^{161}	2^{178}
sntrup4591761 (C)	2265	8.718s	71024	0	?	2^{248}
NTRU-HRSS17 (C)	2416	7.814s	11956	0	2^{123}	2^{136}
NewHope1024-CCA	4032	0.264s	12912	2^{-216}	2^{233}	?
SIKEp751 (C)	1160	685.9s	19112	0	2^{124}	2^{186}

Conclusions

- **Round5** is a merger of two NIST Post-Quantum Competition project KEM & Public Key Encryption algorithms: Hila5 (from me) and Round2 (mainly Philips).
- **GLWR**. Based on (Generalized) Learning With Rounding – no need for random numbers or random samplers, helps to shrink message sizes.
- **Large design space** of n (dimension), p, q, t (moduli - just bit masking), h (ternary secret weight), f (error correction), etc. compared to Ring-LWE or Module-LWE candidates. This helped us to **carefully optimize the parameters**.
- **Bandwidth.** Key sizes and message expansion of Round5 are the smallest among all NIST candidates suitable for embedded use.
- **Performance.** Even the portable C implementation outperforms most assembler-optimized schemes on embedded and PC. HW profile is also good.

Thank You!