ThreeBears

(Round 2)

Carl A. Miller

NIST Computer Security Division

July 23, 2019

NIST PQC Seminar (not for public distribution)

Learning With Errors (LWE) encryption

CRYSTALS-KYBER
FrodoKEM
LAC
NewHope
NTRU
NTRU Prime
Round5
SABER
Three Bears

ThreeBears is a key encapsulation scheme based on **Integer Module-LWE.**

High-Level View

Generic LWE Key Exchange

Alice chooses large n, and a Gaussian noise vector a in $(\mathbf{Z}/n\mathbf{Z})^r$.
Alice publicizes a uniformly random $r x r$ matrix M, and sends
A:= (Ma + Gaussian noise) to Bob.
(The LWE assumption implies that it's hard to recover a from A.)

a, M

A ($\approx$ Ma), M

Generic LWE Key Exchange

Bob chooses a Gaussian noise vector b and publicizes
$B := M^T b$ + Gaussian noise.
Alice and Bob can now both approximately compute $b^T M a$. This info can be used to share a small # of secret bits.

a, M
B ($\approx M^T b$)

A ($\approx Ma$), M, b

Generic LWE Key Exchange

How can this be optimized?

- Use matrices (possibly w/ algebraic structure) in place of *a*, *b*.
- **Use a huge modulus.**

a, M

B ($\approx$ M^Tb)

A ($\approx$ Ma), M, b

Arithmetic in ThreeBears

Consider the ring **Z** / N **Z**, where

$$N = 2^{3120} - 2^{1560} - 1.$$

Express elements of N in binary, 10 digits at a time.

z = 0100101011
1100011011
1011101101
…

Say that an element t is "short" if it can be expressed as

$$t = \Sigma_k \, c_k \, 2^{10\,k}$$

where $\Sigma_k \, |c_k|$ is not very large.

By construction, if s and t are short, then (st) is short.

ThreeBears Key Exchange

In Three Bears, M is a small (up to 4x4) matrix with entries from **Z** / N **Z**.(*)
Alice and Bob use random "short" vectors instead of Gaussian vectors to disguise their operations.

(*): With a modified multiplication operation.

a, M
B ($\approx$ M$^T b$)

A ($\approx$ Ma), M, b

ThreeBears Key Exchange

They end up with approximations to $b^{\mathsf{T}}\mathrm{M}a$ that differ only by "short" elements. That allows them to derive a large number of shared secret bits.

The author mentions that his design is based on KYBER.

a, M

B ($\approx \mathrm{M}^{\mathsf{T}}b$)

A ($\approx \mathrm{M}a$), M, b

Choices & Differences

I-MLWE instead of RLWE or MLWE

"We expected [I-LWE] to be strictly worse than polynomial MLWE, and thus not worthy of a NIST submission. But in fact, I-MLWE gives a range of desirable parameter sets which are comparable to polynomial MLWE in efficiency, ease of implementation, and estimated security."

(Security proof depends on the hardness of I-MLWE – which may be reducible to hardness of MLWE?)

a, M

B ($\approx \mathrm{M}^{\mathrm{T}}b$)

A ($\approx \mathrm{M}a$), M, b

The Modulus

Why $N = 2^{3120} - 2^{1560} - 1$? Apparently because:

- It's a sum/difference of a few powers of 2.
- It's prime.
- It's big enough to encode a 256-bit key.

a, M

B ($\approx$ M^{T}b)

A ($\approx$ Ma), M, b

The Multiplication Operation

Instead of std. multiplication in $\mathbf{Z} / N\,\mathbf{Z}$, the authors use:

$$x * y = xy(2^{1560}-1) \mod N.$$

(The rationale is that if x and y are short, this keeps x*y shorter.)

a, M

B ($\approx$ $M^T b$)

A ($\approx$ Ma), M, b

The Noise Distribution

The individual "digits" of the short noise vectors are chosen as shown. This achieves a particular distribution (not obviously Gaussian) with target variance σ^2.

Function $\texttt{noise}_p(\text{seed}, i)$ **is**

input : Purpose p; seed whose length depends on purpose; index i

require: σ^2 must be either $\begin{cases} \text{in } [0..\frac{1}{2}] \text{ and divisible by } \frac{1}{128} \\ \text{in } [\frac{1}{2}..1] \text{ and divisible by } \frac{1}{32} \\ \text{in } [1..\frac{3}{2}] \text{ and divisible by } \frac{1}{8} \\ \text{exactly } 2 \end{cases}$

output : Noise sample modulo N

$B \leftarrow H_p(\text{seed} \mid\mid [\![i]\!],\ D)$;

for $j = 0$ **to** $D - 1$ **do**

  `// Convert each byte to a digit with var` σ^2

  sample $\leftarrow B_j$;

  $\text{digit}_j \leftarrow 0$;

  for $k = 0$ **to** $\lceil 2 \cdot \sigma^2 \rceil - 1$ **do**

    $v \leftarrow 64 \cdot \min(1,\ 2\sigma^2 - k)$;

    $\text{digit}_j \leftarrow \text{digit}_j + \left\lfloor \frac{\text{sample}+v}{256} \right\rfloor + \left\lfloor \frac{\text{sample}-v}{256} \right\rfloor$;

    sample $\leftarrow$ sample $\cdot\ 4 \bmod 256$;

  end

end

return $\sum_{j=0}^{D-1} \text{digit}_j \cdot x^j \bmod N$

end

Error-Correcting Code

The basic approach to key exchange is this: Bob generates secret random bits $s_1\ s_2\ \ldots\ s_{k,}$ adds them one at a time to the most significant bits of various 10-digit blocks from his estimate for $b^{\mathrm{T}}\mathrm{M}a$, and then transmits the result.

However, before doing this he applies the "Melas" encoding scheme to s.

a, M

B ($\approx \mathrm{M}^{\mathrm{T}}b$)

A ($\approx \mathrm{M}a$), M, b

Error-Correcting Code

The Melas encoding scheme adds 18 bits and corrects up to 2 errors.
This addresses failures that can occur from the accumulation of noise.

"Our Melas implementation has small code and memory requirements, runs in constant time, and is so fast that its runtime is almost negligible. Its downsides are increased complexity, and a correspondingly wider attack surface for side-channel and fault attacks."

a, M
B ($\approx$ M$^{\mathrm{T}}b$)

A ($\approx$ Ma), M, b

Performance

Speed (in cycles)

System	CPA-secure KeyGen	CPA-secure Enc	CPA-secure Dec	CCA-secure KeyGen	CCA-secure Enc	CCA-secure Dec
Skylake (high speed)						
BabyBear	41k	62k	28k	41k	60k	101k
MamaBear	84k	103k	34k	79k	96k	156k
PapaBear	124k	153k	40k	118k	145k	211k
Cortex-A53						
BabyBear	153k	211k	80k	154k	210k	351k
MamaBear	302k	377k	111k	297k	369k	566k
PapaBear	500k	594k	141k	492k	582k	840k
Cortex-A8						
BabyBear	344k	501k	176k	345k	495k	810k
MamaBear	729k	943k	260k	720k	931k	1379k
PapaBear	1234k	1511k	319k	1225k	1502k	2134k
Cortex-M4 (high speed)						
BabyBear	644k	841k	273k	644k	824k	1299k
MamaBear	1266k	1521k	381k	1257k	1494k	2174k
PapaBear	2095k	2409k	488k	2082k	2378k	3272k
Cortex-M4 (low memory)						
BabyBear	744k	1039k	273k	744k	1022k	1495k
MamaBear	1564k	1967k	381k	1548k	1929k	2609k
PapaBear	2691k	3201k	488k	2663k	3150k	4044k

Space

System	Private key	Public key	Capsule
BabyBear	40	804	917
MamaBear	40	1194	1307
PapaBear	40	1584	1697

Table 10: ThreeBears object sizes in bytes

They also give numbers for code size.

Memory

System	CPA-secure Keygen	CPA-secure Enc	CPA-secure Dec	CCA-secure Keygen	CCA-secure Enc	CCA-secure Dec
Skylake (high speed)						
BabyBear	6216	6632	4232	6216	6632	8184
MamaBear	9112	9528	4632	9112	9560	11512
PapaBear	12856	13272	5048	12856	13304	15672
Skylake (low memory)						
All instances	2392	2424	2168	2392	2424	3080
Cortex-M4 (high speed)						
BabyBear	2760	2832	2080	2760	2832	4944
MamaBear	3256	3312	2080	3256	3320	5904
PapaBear	3736	3800	2080	3736	3800	6864
Cortex-M4 (low memory)						
All instances	2288	2352	2080	2288	2352	3024

Table 12: ThreeBears memory usage bytes, excluding input and output.

https://eprint.iacr.org/2019/844.pdf numerically compares ThreeBears & other candidates in a resource-limited environment.

Failure Probability

System	d	cca	σ^2	Failure	Lattice security Classical	Quantum	Class
BabyBear	2	0	1	$\approx 2^{-58}$	168	153	II
		1	9/16	$< 2^{-156}$	154	140	II
MamaBear	3	0	7/8	$\approx 2^{-51}$	262	238	V
		1	13/32	$< 2^{-206}$	235	213	IV
PapaBear	4	0	3/4	$\approx 2^{-52}$	351	318	V
		1	5/16	$< 2^{-256}$	314	280	V

Table 2: ThreeBears recommended parameters. Security levels are given as the $\log_2$ of the estimated work to break the system using a lattice or chosen-ciphertext attack on a quantum computer.

ThreeBears

(Round 2)

Carl A. Miller
NIST Computer Security Division

July 23, 2019

NIST PQC Seminar (not for public distribution)