Module-LWE key exchange and encryption: The three bears

Mike Hamburg[*]

Draft 1; expecting to add Hart Montgomery as co-author

June 18, 2017

Abstract

We propose a new post-quantum key exchange algorithm based on the module learning with errors (mLWE) problem. Our ThreeBears algorithm is simple and performant, but our main goal is to suggest mLWE over a pseudo-Mersenne field instead of a polynomial ring. We also show how to build a public-key encryption system from the key exchange algorithm.

1 Introduction

All widely-deployed key exchange and public-key encryption algorithms are threatened by the possibility of a quantum computer powerful enough to run Shor's algorithm [28]. Consequently, there is a growing interest in developing a suite of "post-quantum" algorithms which would resist attack by these computers [21]. The most common approaches to addressing this threat rely on the hardness of lattice problems, including variants such as learning with errors (LWE) [24], ring learning with errors (rLWE) [20], and module learning with errors (mLWE) [19].

Overall, rLWE-based schemes [15, 14, 2] tend to be faster and have smaller public keys and ciphertexts than those based on classical LWE [6].

[*]Rambus Security Division

This is enabled by the extra structure that the ring provides, but there is a lingering concern that this structure will enable new attacks [23]. This concern has led to proposals for rings with less structure [4], and for rings to be replaced with modules when the full ring structure is not necessary [3].

1.1 Our contribution

Here we propose a new cryptosystem based on mLWE, where the underlying ring is the integers modulo a pseudo-Mersenne number. This opens up a new set of implementation options, and may have better (or worse!) security properties than a polynomial ring.

Our cryptosystem is called ThreeBears, because its pseudo-Mersenne prime has the same shape as the one in Ed448-Goldilocks [12].

There are few systems based on lattices modulo Mersenne and pseudo-Mersenne numbers. Though previous proposals [1] have met with limited success [5], we believe that combining pseudo-Mersenne lattices with state-of-the-art key exchange protocols will at least be worth analyzing.

We also further explore Saarinen's error correction trick [26, 25]. For unauthenticated key exchange, we have tuned the parameters for a small signal-to-noise ratio in order to minimize bandwidth costs and maximize security. We did this at the cost of key exchange failures, which occur with probability around 2^{-56} for our weakest recommendation.

But for public-key encryption, failures may lead to a chosen-ciphertext attack [16]. While we could use a larger signal-to-noise ratio, we instead use error-correcting codes. With a code that corrects up to two errors, we can roughly cube the failure probability. We then estimate the complexity of attacks against the corrected key, and show that they are in line with security estimates for lattice attacks. This analysis is mostly independent of the pseudo-Mersenne lattice aspect of the design, and so may be of independent interest.

2 Ring choice

The choice of ring is one of the most important decisions when designing a ring-LWE-based system. Here we review polynomial rings, and then examine our choice, pseudo-Mersenne rings.

2.1 Polynomial rings

Most previous systems use polynomial rings of the form

$$R := (\mathbb{Z}/q\mathbb{Z})[x]/P(x)$$

for some integer q and monic polynomial P. They also specify one or more noise distributions χ_i over R. Typically the noise is defined as

$$\chi := \sum_{i=0}^{\deg(P)-1} \epsilon_i x^i \quad \text{where} \quad \epsilon_i \leftarrow \psi$$

where ψ may be a binomial [2], uniform or discrete Gaussian [7] distribution. Alternatively, the noise may be chosen to have a fixed Hamming weight [4], or may be a consequence of rounding [9]. In any case, for decryption to work we will need $\chi_i \cdot \chi_j$ to be "small" over R, i.e. for its coefficients to have standard deviation much less than q. This condition usually implies that the polynomial P is sparse with small coefficients.

2.2 Rings modulo pseudo-Mersenne numbers

Here we take a slightly different approach. We choose an integer x and a polynomial P, and set

$$N := P(x) \quad \text{and} \quad R := \mathbb{Z}/N\mathbb{Z}$$

That is, R is the ring of integers modulo a Mersenne number or pseudo-Mersenne number N. The integer x plays the role of both the modulus q and the formal variable x in a polynomial ring. We can then choose the noise in the analogous way:

$$\chi := \sum_{i=0}^{\deg(P)-1} \epsilon_i x^i \quad \text{where} \quad \epsilon_i \leftarrow \psi$$

or we can use a fixed Hamming weight or even rounding. Again, limiting errors generally requires P to be sparse with small coefficients.

For our main recommendations, we let $x = 2^{10}$ and $P = x^{312} - x^{156} - 1$, resulting in the "golden Solinas" pseudo-Mersenne prime

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

For some of our toy instances, we instead let $x = 2^8$ and $P = x^{270} - x^{135} - 1$, resulting in $N = 2^{2160} - 2^{1080} - 1$. Both of these N values are prime, which rules out attacks based on subrings. For an analysis of some other options, see Appendix A.

In a break with tradition, $\deg(P)$ is neither prime nor a prime power. In cyclotomic fields, using a prime or prime power mitigates weaknesses based on subrings of R. But when N is prime there are no subrings of $\mathbb{Z}/N\mathbb{Z}$.

2.3 Pros and cons of pseduo-Mersenne rings

Both polynomial rings and pseudo-Mersenne rings have their advantages. Here is a brief comparison.

Security The security of polynomial rings has been considered for longer, which is reassuring. However, pseudo-Mersenne rings have a different ring structure, and when N is prime they have no subrings. This may improve or weaken security.

Multiplication algorithms Some polynomial rings support fast multiplication based on the number-theoretic transform (NTT). Since the NTT can be performed in place, these rings have an intrinsic memory savings. Elements can even be sampled and sent in the NTT domain [2], though this constrains implementations and complicates the specification. Other systems avoid the extra structure required to support the NTT — that P is cyclotomic and splits mod q — out of concerns that it may lead to security problems [4].

Pseudo-Mersenne numbers support multiplication algorithms such as Karatsuba-Ofman [18] or Granger-Moss [10], as well as Solinas reduction [29].

These algorithms are faster than naïve "schoolbook" algorithm, but they are not as efficient as NTT-based multiplication. Usually big-number multiplication cannot be performed in-place.

Processor and hardware support Polynomial rings typically have 10- to 14-bit q, so they do arithmetic with 16-bit `short`s and do not require multi-precision arithmetic. This means they work well both on tiny machines with tiny registers and on large machines with vector units. The small coefficients make it easier to mitigate problems with multi-precision arithmetic encountered on (for example) the ARM Cortex-M0 and M3. It also may make it easier to implement these designs in new cryptographic hardware.

A large prime field is implemented internally with digits or "limbs" of an architecture-dependent size. For example, a 64-bit machine could use either a full 64 bits per limb, or drop to e.g. 60 bits for improved carry handling. Either way, the relatively dense packing of limbs makes storage of elements more memory-efficient, offsetting the disadvantage of out-of-place multiplication. It also means that these designs can take advantage of existing cryptographic hardware and software libraries, which typically support multi-precision arithmetic.

Powers of 2 Making x or q a power of 2 makes it easier to convert to and from wire formats, and simplifies reconciliation. It also simplifies uniform sampling from the ring, which meaningfully reduces runtime. For a pseudo-Mersenne number, making x a power of 2 is the logical choice. For polynomial rings, NTRU sets q to a power of 2. But most systems instead use a prime q, either to avoid subrings or to leverage fast NTT-based multiplication [2, 3].

Error amplification Cyclotomic rings tend to have smaller error amplification than other rings. This results in slightly better performance and security. This advantage is not shared by non-cyclotomic polynomial rings, nor by our pseudo-Mersenne fields.

Division Division in a polynomial ring is faster than in a large prime-order field, especially if $P(x)$ splits over the base field. Division is used in NTRU-like protocols [15, 4] but not in variants of the Ding [17] and Peikert [22] protocols. We are proposing a Ding-like protocol, so we don't need division.

Summary Overall, we believe that pseudo-Mersenne rings are about as suitable for this task as polynomial rings. But these rings have received less attention for lattice problems. We are proposing ThreeBears as a step to correcting this gap.

2.4 Modules

It is most convenient to exchange keys whose bit length is equal to the dimension of the ring. Post-quantum systems must contend with Grover's algorithm [11], so a 256-bit key is appropriate for high-security systems.

Accordingly, we have chosen rings with dimension slightly more than 256. However, this is too small a dimension to effectively resist lattice reduction algorithms such as BKZ [27, 8]; for this, it seems that dimension 500-1000 is required. We address this problem by using the vector space R^d for some small dimension d. In our recommended parameters, d is between 1 and 4.

2.5 Error distribution

We will make our error distribution on the ring by applying a simple distribution ψ_{σ^2} to each coordinate. When $\sigma^2 \leq 2$, let

$$\psi_{\sigma^2} := \begin{cases} -1 & \text{with probability } \sigma^2/2 \\ 0 & \text{with probability } 1-\sigma^2 \\ +1 & \text{with probability } \sigma^2/2 \end{cases}$$

When $\sigma^2 = k/2^n \leq 1/2$, our ψ_{σ^2} can be sampled as follows:

$$\begin{aligned} r_{i,j} &\overset{R}{\leftarrow} [0, 2^{n+1}) \\ \epsilon_{i,j} &\leftarrow \lfloor (r+k)/2^{n+1} \rfloor + \lfloor (r-k)/2^{n+1} \rfloor \end{aligned}$$

When $\sigma^2 > 2$, instead let

$$\psi_{\sigma^2} := \psi_{1/2} + \psi_{\sigma^2 - 1/2}$$

In both cases, ψ_{σ^2} has mean 0 and variance σ^2.

Over the module, we will use the error distribution

$$\chi_{\sigma^2} := \left[\sum_{i=0}^{\deg(P)-1} \epsilon_{i,j} \cdot x^i\right]_{j=0}^{d-1} \in R^d \quad \text{where} \quad \epsilon_{i,j} \leftarrow \psi_{\sigma^2} \text{ independently}$$

That is, we will apply ψ_{σ^2} independently to each limb over the ring, independently for each of the d ring elements that make up a module element.

[[TODO: Shaped noise? eg less noise at the lsb and msb, more in the middle? Noise that affects more than one coefficient at once? Gaussian noise??? Need Hart's expertise.]]

2.6 Clarifier

For non-cyclotomic rings, it turns out that multiplying two samples of the error distribution produces a larger combined error than necessary. We will mitigate this by applying a *clarifier.*

Let clar $\in R^*$ be a value which minimizes the variance of the coefficients of

$$\text{clar} \cdot \chi_{\sigma^2}^\top \cdot \chi_{\sigma^2}$$

If $R = \mathbb{Z}/(\phi^2 - \phi - 1)\mathbb{Z}$ for some integer ϕ, the optimal clarifier is $1/\phi = \phi - 1$. That is, when $R = \mathbb{Z}/(2^{3120} - 2^{1560} - 1)\mathbb{Z}$, we set clar $= 2^{1560} - 1$. Likewise, for $R = \mathbb{Z}/(2^{2160} - 2^{1080} - 1)\mathbb{Z}$, we set clar $= 2^{1080} - 1$.

2.7 Uniform distribution from seed

We will also need to sample almost-uniformly from matrices in $R^{d\times d}$ with a seed string $s \in \{0,1\}^\ell$. Let

$$U_1 :: \{0,1\}^\ell \times [0,d)^2 \to R$$

be a function of a seed s and small indices i, j. Then

$$U(s) := \begin{pmatrix} U_1(s,d) & \dots & U(s,0,d-1) \\ \vdots & \ddots & \vdots \\ U(s,d-1,0) & \dots & U(s,d-1,d-1) \end{pmatrix} \in R^{d\times d}$$

Concretely, a conservative choice is

$$U_1(s, i, j) := \text{decode}(\text{SHAKE256}(s||[d, i, j];\ \text{length} = \log_2 N))$$

On platforms that accelerate AES in hardware, a faster choice is

$$U_1(s, i, j) := \text{decode}(\text{AES256-CTR}(s; \text{nonce}; \text{pt}))$$
$$\text{where nonce} = [d, i, j, 0, \ldots, 0] \text{ and pt} = \log_2 N \text{ bits of zeros}$$

For lightweight protocols, SHAKE256 can be replaced with Strobe [13]'s PRF operation.

3 Key exchange protocol

Pseudo-Mersenne rings are suitable for most of the same protocols that polynomial rings are used for. As a start, we will describe a Ding-like key exchange protocol [17], as shown in Figure 1.

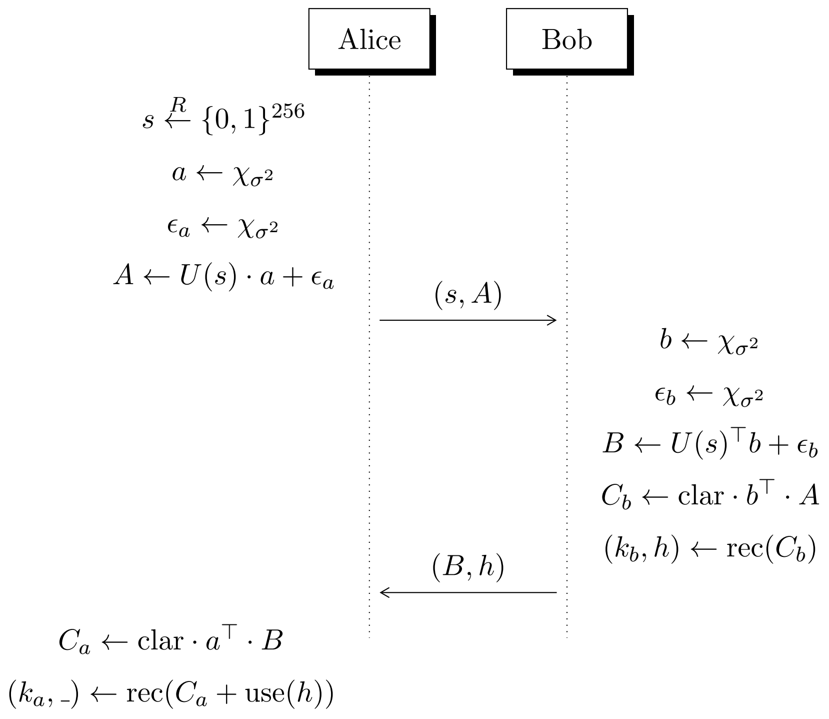
Alice
Bob
$s \xleftarrow{R} \{0,1\}^{256}$
$a \leftarrow \chi_{\sigma^2}$
$\epsilon_a \leftarrow \chi_{\sigma^2}$
$A \leftarrow U(s) \cdot a + \epsilon_a$
(s, A)
$b \leftarrow \chi_{\sigma^2}$
$\epsilon_b \leftarrow \chi_{\sigma^2}$
$B \leftarrow U(s)^\top b + \epsilon_b$
$C_b \leftarrow \text{clar} \cdot b^\top \cdot A$
$(k_b, h) \leftarrow \text{rec}(C_b)$
(B, h)
$C_a \leftarrow \text{clar} \cdot a^\top \cdot B$
$(k_a, \_) \leftarrow \text{rec}(C_a + \text{use}(h))$

Figure 1: Ding-like key exchange protocol

3.1 Alice's message

Suppose Alice and Bob want to compute a shared secret. First Alice chooses a random seed $s \overset{R}{\leftarrow} \{0,1\}^{256}$, and then chooses $a \leftarrow \chi_{\sigma^2}$ and $\epsilon_a \leftarrow \chi_{\sigma^2}$. Alice sends to Bob the values

$$(s, \quad A := U(s) \cdot a + \epsilon_a)$$

3.2 Bob's message and shared secret

Bob chooses his own secrets $b \leftarrow \chi_{\sigma^2}$ and $\epsilon_b \leftarrow \chi_{\sigma^2}$, and computes

$$B := U(s)^\top \cdot b + \epsilon_b \text{ and } C_b := \text{clar} \cdot b^\top \cdot A$$

Bob computes $\text{rec}(C_b)$ as follows. He writes C_b as

$$C_b = \sum_{i=0}^{\deg(P)-1} c_i \cdot x^i$$

and set $(K_b)_i$ to the top bit of c_i (with significance $x/2$), and h_i to its second-top bit (with significance $x/4$). Bob's copy of the shared secret is then $K_b := [[(K_b)_i]]$, and his reconciliation value is $h := [[h_i]]$.

In practice we will truncate[1] the help and key to a convenient length, either 256 bits or 256 plus error correction values (see Section 4).

Bob sends (B, h) to Alice and outputs the key k_b.

3.3 Alice's shared secret

Alice likewise computes

$$C_a := a^\top \cdot B$$

She expands the help value to

$$\text{use}(h) := \sum_{i=0}^{\deg(P)-1} \frac{1 - 2 \cdot h_i}{8} \cdot x^{i+1}$$

[1]Because the middle coefficients have higher error probabilities than the outside ones, it may be beneficial to remove those coefficients first.

She computes

$$(K_a, h_a) := \text{rec}(C_a + \text{use}(h))$$

just as Bob did. She throws away h_a and outputs the key K_a.

Alice's key K_a agrees with Bob's key K_b with high probability. See Appendix C for correctness criteria, and Appendix D for an analysis of the failure probability.

4 Forward error correction

As with any LWE-based cryptosystem, it is easy to trade between failure probability, dimension and security. We chose parameters that put the failure probability in an appropriate range for key exchange, around 2^{-56} for BabyBear[2]. For CCA-secure encryption, this failure rate is not acceptable, because an attacker may learn information about the private key every time there is a decryption error. To fix this, we follow Saarinen's tack of using error correcting codes for forward error correction (FEC) [26, 25].

We use a Melas-style BCH$(511, 493, 5)$ code [?], which can correct 2 errors in up to 511 bits at the cost of 18 bits of overhead. Since the error correction is sent in the clear, it gives 18 bits of information about the key, so the key must be 18 bytes longer. For simplicity, we round both of these overheads up to 3 bytes.

We describe the implementation of this Melas code in more detail in Appendix B. Since our larger N gives a 312-bit key, it has room for up to 56 bits of error correction. A larger BCH code would certainly work, but might be overly complex, and would gain little in security. A code that supports soft decoding would also be an interesting choice. As far as we know, it is an open problem to optimize the error correction for LWE systems.

We analyzed the effect of forward error correction on failure probability for both key exchange and encryption. See Appendix D for details.

[2]A failure probability of 10^{-5} per year is more than 2^{-52} per millisecond.

5 CCA-secure encryption

[[**TODO: should we derive bigger keys to prevent a quantum time-data tradeoff attack???**]]

In the (quantum) random oracle model, we can convert this key exchange into a CCA-secure public-key encryption algorithm by using a variant of the Targhi-Unruh conversion [30]. We will need to create algorithms **Keygen**, **Encrypt** and **Decrypt**. To do this, we define an algorithm $\hat{\chi}_{\sigma^2}(k)$ that samples χ_{σ^2} from a seed k using a random oracle hash function H.[[**TODO: come back to that**]]

Keygen To generate a keypair, Alice chooses a random 256-bit private key α uniformly at random. She computes

$$\begin{aligned} s &\leftarrow H(\alpha, 0) : 256 \text{ bits} \\ a &\leftarrow \hat{\chi}_{\sigma^2}(H(\alpha, 1)) \\ \epsilon_a &\leftarrow \hat{\chi}_{\sigma^2}(H(\alpha, 2)) \\ A &\leftarrow U(s) \cdot a + \epsilon_a \end{aligned}$$

Alice's public key is then

$$\text{pk}_a := (s, A)$$

Encrypt To encrypt a message m with Alice's public key, Bob first chooses a random 256-bit key β uniformly at random. He computes

$$\delta \leftarrow H(\text{pk}_a, \beta, 0)$$

which is required for the proof of the Targhi-Unruh conversion. He then proceeds as in the KEM, except that he samples χ deterministically from his seed β:

$$\begin{aligned} b &\leftarrow \hat{\chi}_{\sigma^2}(H(\mathrm{pk}_a, \beta, 1)) \\ \epsilon_b &\leftarrow \hat{\chi}_{\sigma^2}(H(\mathrm{pk}_a, \beta, 2)) \\ B &\leftarrow U^\top(s) \cdot b + \epsilon_b \\ C_b &\leftarrow \mathrm{clar} \cdot b^\top \cdot A \\ (k_b, h) &\leftarrow \mathrm{rec}(C_b) \end{aligned}$$

Bob computes forward error correction on the key (which also means that the key must be slightly longer than for encryption):

$$f \leftarrow \mathrm{fec\_compute}(k_b)$$

He then encrypts his seed β under the derived key so that Alice can verify that he has been honest:

$$\gamma \leftarrow H(k_b, 0) \oplus \beta$$

Finally, he encrypts the message using $H(k_b, 1)$ using some CCA-secure symmetric encryption scheme $\mathrm{Enc}(key; m)$:

$$c \leftarrow \mathrm{Enc}(H(k_b, 1); m)$$

The ciphertext is then

$$(B, h, f, \gamma, \delta, c)$$

Decrypt To decrypt a ciphertext $(B, h, f, \gamma, \delta, c)$, Alice first recomputes her public key and a. She then computes

$$\begin{aligned} C_a &\leftarrow \mathrm{clar} \cdot a^\top \cdot B \\ (\tilde{k}_a, \_) &\leftarrow \mathrm{rec}(C_a + \mathrm{use}(h)) \\ k_a &\leftarrow \mathrm{fec\_correct}(\tilde{k}_a, f) \\ \beta &\leftarrow H(k_a, 0) \oplus \gamma \end{aligned}$$

Alice then recomputes $(B, h, f, \gamma, \delta)$ using **Encrypt** to make sure that Bob encrypted properly; if the result is different, she rejects the message. Finally, she recovers the message as

$$m \leftarrow \text{Enc}(H(k_a, 1); c)$$

[[**TODO: Analysis. Do we need longer keys? Figure out the right place(s) to hash pka.**]]

5.1 Recommended and toy parameters

We chose three recommended sets of parameters: BabyBear, MamaBear and PapaBear. These are designed with estimated security of 128, 192 and 256 bits, respectively, against known quantum attacks; we expect this to drop as more work is done in attacking quantum systems. We also chose three "toy" sets, GummyBear, TeddyBear and DropBear, intended to stimulate cryptanalysis. Our primary recommendation is MamaBear. We compare our parameters to related work in Table 1.

We tuned the variance of the noise in these systems by intervals of 1/32 in order to balance active vs. passive attacks. Once the attacks are better understood, it may be worth retuning them, perhaps to a finer granularity.

[[**TODO: kyber etc? NTRU Prime? Lizard?**]] [25]

5.2 Security analysis

We evaluated our system against five different security metrics, and compared to claims in related work.

The first three, labeled "C", "Q" and "P", are core SVP hardness against primal or dual lattice-reduction attacks against the public key. These correspond to NewHope's core hardness against "known classical", "known quantum" and "best possible" attacks, and give optimistic (from the attacker's point of view) estimates of the difficulty of recovering the private key using BKZ [27, 8]. We estimated this using NewHope's BKZ 2.0 parameter estimation script [2]. Note that NTRU Prime's estimates were done using a more realistic model of BKZ costs, which leads to a higher security estimate. [[**TODO: need Hart's expertise here**]]

System	Toy?	Ref	d	d_{total}	σ^2	q
BabyBear		This paper	2	624	11/32	1024
MamaBear		This paper	3	936	7/32	1024
PapaBear		This paper	4	1248	5/32	1024
GummyBear	Y	This paper	1	270	1/8	256
TeddyBear	Y	This paper	1	390	1/8	256
DropBear	Y	This paper	2	540	3/32	256
NewHope		[2]		1024	8	12289
JarJar	Y	[2]		512	12	12289
trunc8		[26]		512	23.6	12289
Hila5		[25]		1024	8	12289
NTRUEncrypt		[15]		743	$\approx 2/3$	2048
NTRU Prime		[4]		739	0.28	9829

Table 1: Parameters for ThreeBears and related work.

The next two, labeled "F" and "G" for Failure and Failure+Grover, are chosen-ciphertext attacks on encryption with a long-term public/private key pair. The Failure attack is the effort for a classical attacker to send random ciphertexts until one fails to decrypt, which will give attacker information about the private key. We have not analyzed the number of failures required to recover the key, but instead show the effort required to produce a single failure.

These attack model are additionally optimistic from the attacker's point of view, because they measure effort as work divided by success probability and do not assign a cost to chosen-ciphertext queries. Per success, therefore, the attacks require far in excess of NIST's recommended 2^{64} chosen ciphertexts.

The Failure+Grover attack is the same, but using Grover's algorithm on a quantum computer to find chosen ciphertexts with large norm. The classical version of this attack can be used to reduce the number of decryption queries, but not the total effort, so it is modeled by Failure alone. Because the public key is hashed in with the seed for the ciphertext, this attack still only targets one public/private key pair. This attack is analyzed in

	lattice security			failure security			Message bytes		
System	C	Q	P	Bit	F	G	Pub	KEM	Enc
BabyBear	142	128	100	56	157	146	812	812	882
MamaBear	214	194	151	86	235	215	1202	1202	1272
PapaBear	284	258	201	117	315	284	1592	1592	1662
GummyBear	51	46	36	46	117	102	302	302	372
TeddyBear	82	75	58	36	95	85	422	422	492
DropBear	116	105	82	42	113	101	572	572	642
JarJar	131	118	92	55	-	-	928	1024	-
NewHope	281	255	199	61	-	-	1824	2048	-
trunc8	141	131	102	13	45	-	1024	1024	-
Hila5	281	255	199	27	135	-	1824	2012	2012
NTRUEncrypt	176	159	125	112	112	-	1022	1022	1022
NTRU Prime	215	-	128	∞	∞	∞	1232	1141	1141

Table 2: Security and message sizes for ThreeBears and related work. Security estimates are log base 2 of the conservatively estimated attack effort. The "bit" column is the estimated $-\log_2$ probability of a single-bit failure before error correction — or equivalently, of a failure in key exchange — and does not represent an attack.

Appendix D.

Of these attacks, we have tuned our parameters for security against "Q" and Failure+Grover, because these correspond to the hard steps of known quantum attacks. We are more concerned about passive lattice reduction attacks than failure attacks, because the failure attacks given here are completely infeasible with a realistic number of chosen ciphertexts.

The failure attacks have room for improvement. It may be possible to design a "fuzzy Grover" algorithm that preferentially samples ciphertexts with a high failure probability, and would outperform a straightforward Grover attack in this scenario. It may also be possible to create more correlation between bit failures, which our analysis does not account for. On the other hand, we expect that a more realistic cost model would greatly increase the nominal attack effort. In any case, we have left a much larger security margin on the Failure+Grover attack than we would have if it were fully developed.

Because NTRU Prime is immune to failures, it is also immune to these failure attacks. We did not attempt to evaluate the Failure+Grover attack on other systems, but it is probably worth evaluating for Hila5.

The sizes of public keys and KEM messages or ciphertexts is a meaningful obstacle in deploying post-quantum cryptography, and must be traded off against security. We therefore compare these metrics as well.

6 Performance

We created a reference implementation of ThreeBears in C, optimized for simplicity and memory consumption. The reference code contains no processor-specific optimizations, but it can take advantage of $64 \times 64 \rightarrow 128$-bit multiplication when the compiler and CPU support them. Our arithmetic code uses one level of Karatsuba multiplication, because we cribbed its arithmetic code from Ed448-Goldilocks [12]. More levels would be faster, but we didn't bother in the initial implementation since the bottleneck is actually SHAKE.

We compiled our code with `clang-3.8 -Os` and ran it on an Intel Core i3-6100U Skylake CPU at 2.3 GHz. On this processor, it took about 410k cycles to perform both the client and server side of the key exchange (i.e. 219kcy for each side), including for both Alice and Bob to compute K_{hashed}. The multiplications modulo N took about 30% of the compute time, with most of the balance being SHAKE. The error correction took only a few thousand cycles.

System	Skylake
BabyBear	127k
MamaBear	219k
PapaBear	333k

Table 3: Performance in cycles, including error correction.

[[**TODO: Optimized numbers; ARM; reflect updates in sampling and clarification; consider AES sampler**]]

6.1 Intellectual property

The authors are not aware of any patents which apply to this work. Do not take this as a guarantee that there are no such patents, as cryptography is a patent minefield and company policy prohibits looking for the mines.

The authors' institutions intend for ThreeBears to be an open standard. [[**TODO: Statement from legal about how we won't patent it, but (depending what legal says) we might patent DPA countermeasures or something.**]]

7 Future work

We plan to formally specify ThreeBears, or some closely related scheme, in order to submit it to the NIST post-quantum cryptography project [21]. We also plan to improve the analysis of its security, and possibly to improve the error correcting code or noise distributions. We welcome the publication of cryptanalysis, implementations, and systems derived from ThreeBears.

8 Conclusion

In this paper, we presented ThreeBears, a relatively simple instantiation of module LWE based on pseudo-Mersenne numbers. This system provides an alternative to polynomial rings for ring- and module-LWE instances. It may be used exchange or public-key encryption, and we hope that it is able to resist both classical and quantum attack in these settings.

References

[1] Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Miklos Santha. A new public-key cryptosystem via mersenne numbers. Cryptology

ePrint Archive, Report 2017/481, 2017. `http://eprint.iacr.org/2017/481`.

[2] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. In *USENIX Security 2016*, 2016. `http://eprint.iacr.org/2015/1092`.

[3] Shi Bai, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. From NewHope to Kyber, 2017. `https://cryptojedi.org/peter/data/inria-20170411.pdf`.

[4] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime. Cryptology ePrint Archive, Report 2016/461, 2016. `http://eprint.iacr.org/2016/461`.

[5] Marc Beunardeau, Aisling Connolly, Rmi Graud, and David Naccache. On the hardness of the Mersenne low Hamming ratio assumption. Cryptology ePrint Archive, Report 2017/522, 2017. `http://eprint.iacr.org/2017/522`.

[6] 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 *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1006–1018, New York, NY, USA, 2016. ACM.

[7] 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. *IEEE Security and Privacy*, 2015. `http://eprint.iacr.org/2014/599`.

[8] Yuanmi Chen and Phong Nguyen. Bkz 2.0: Better lattice security estimates. *Advances in Cryptology–ASIACRYPT 2011*, pages 1–20, 2011.

[9] 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. Cryptology ePrint Archive, Report 2016/1126, 2016. `http://eprint.iacr.org/2016/1126`.

[10] Robert Granger and Andrew Moss. Generalised Mersenne numbers revisited. *Mathematics of Computation*, 82(284):2389–2420, 2013.

[11] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[12] Mike Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. `http://eprint.iacr.org/2015/625`.

[13] Mike Hamburg. The STROBE protocol framework. Real World Crypto, 2017. `http://eprint.iacr.org/2017/003`.

[14] Jeff Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing parameters for NTRU-Encrypt. Cryptology ePrint Archive, Report 2015/708, 2015. `http://eprint.iacr.org/2015/708`.

[15] Jeffrey Hoffstein, Jill Pipher, and Joseph Silverman. NTRU: A ring-based public key cryptosystem. *Algorithmic number theory*, pages 267–288, 1998.

[16] Nick Howgrave-Graham, Phong Q Nguyen, David Pointcheval, John Proos, Joseph H Silverman, Ari Singer, and William Whyte. The impact of decryption failures on the security of NTRU encryption. In *Annual International Cryptology Conference*, pages 226–246. Springer, 2003.

[17] Xiaodong Lin Jintai Ding, Xiang Xie. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688, 2012. Also published at EUROCRYPT 2014. `http://eprint.iacr.org/2012/688`.

[18] A Karabutsa and Yu Ofman. Multiplication of many-digital numbers by automatic computers. *Doklady Akademii Nauk SSSR*, 145(2):293, 1962.

[19] Adeline Langlois and Damien Stehle. Worst-case to average-case reductions for module lattices. Cryptology ePrint Archive, Report 2012/090, 2012. `http://eprint.iacr.org/2012/090`.

[20] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):43, 2013.

[21] Dustin Moody, Lily Chen, and Yi-Kai Liu. Post-quantum crypto project, 2016. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/`.

[22] Chris Peikert. Lattice cryptography for the internet, 2014. `http://eprint.iacr.org/2014/070`.

[23] Chris Peikert. How (not) to instantiate ring-lwe. Cryptology ePrint Archive, Report 2016/351, 2016. `http://eprint.iacr.org/2016/351`.

[24] Oded Regev. The learning with errors problem. *Invited survey in CCC*, page 15, 2010.

[25] Markku-Juhani O. Saarinen. On reliability, reconciliation, and error correction in ring-lwe encryption. Cryptology ePrint Archive, Report 2017/424, 2017. `http://eprint.iacr.org/2017/424`.

[26] Markku-Juhani Olavi Saarinen. Ring-LWE ciphertext compression and error correction: Tools for lightweight post-quantum cryptography. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, IoTPTS '17, pages 15–22, New York, NY, USA, 2017. ACM.

[27] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.

[28] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[29] Jerome A. Solinas. Generalized mersenne numbers. Technical report, Waterloo, 1999.

[30] Ehsan Ebrahimi Targhi and Dominique Unruh. Quantum security of the Fujisaki-Okamoto and OAEP transforms. Cryptology ePrint Archive, Report 2015/1210, 2015. `http://eprint.iacr.org/2015/1210`.

A Other ring choices

Readers may be curious why we chose this specific ring

$$R = \mathbb{Z}/N_{\text{bears}}\mathbb{Z} \text{ where } N_{\text{bears}} = 2^{3120} - 2^{1560} - 1$$

Certainly some sort of generalized Mersenne number is required to minimize error amplification, but why this one? We considered rings of a few other shapes, but ultimately settled on R, as this section will explain.

The most obvious choice would be the integers modulo a Mersenne prime, such as $p_{3217} := 2^{3217} - 1$. This prime is conveniently equal to $2^{12\cdot268+1} - 1$, which means that clar $= 2$ would work nicely. However, the error amplification in this ring is higher than in our R, because after clarifying and reducing mod p_{3217} some coefficients will be doubled. This increases the variance they contribute to the failure estimates by a factor of 4, instead of $3/2$ for N_{bears}.

A more general alternative is a cyclotomic field of the form $\mathbb{Z}/\Phi_k(2)\mathbb{Z}$ for some k. Such a field will usually have unacceptable error amplification, but we can lift to $\mathbb{Z}/(2^k \pm 1)\mathbb{Z}$ by choosing a clarifier divisible by $(2^k \pm 1)/\Phi_k(2)$. For example, $\Phi_{2\cdot607\cdot13}(2)$ works [[**TODO: with clarifier...**]]. We did not see an appreciable gain in this approach, though the resulting system is at least mathematically interesting.

A final possibility is a hybrid approach, where instead of $(\mathbb{Z}/q\mathbb{Z})[x]/P(x)$ or $\mathbb{Z}/P(k)\mathbb{Z}$ we choose a ring of the form

$$(\mathbb{Z}/P(k)\mathbb{Z})\ [x]\ /\ (Q(k,x))$$

In other words, we can build a polynomial ring on top of a pseudo-Mersenne field, with multiple coefficients packed into each field element. While this construction gives us many choices, it is also more complex. Since the main goal of ThreeBears is to provide a clean alternative to polynomial rings, this construction wouldn't be as interesting to analyze.

Within Solinas primes, golden-ratio ones such as N_{bears} seem to provide the smallest error amplification and the widest selection of implementation choices. This is why we chose a golden-ratio Solinas prime over other primes such as $2^{12\cdot 256} - 2^{12\cdot 103} - 1$. Within golden-ratio Solinas primes, our choice was driven by need for a digit size of at least 2^{10} with a degree at least $256 + 18$ (for error correction); or at least 2^{12} with degree at least 256 (with no error correction).

Using error correction leads to smaller messages and better performance at a given security level. However, if someone wanted to use a variant ThreeBears without error correction, the best approach would probably be to use $x = 2^{12}$ and $P = 2^{260} - 2^{130} - 1$. This gives the same N, but since noise is smaller and the degree of P is smaller, we need a higher d for the same security level.

B Implementation of the Melas code

Our Melas code is fairly straightforward. We treat the data to be error-corrected as a polynomial in a formal variable t over $GF(2)$, and reduce it modulo the primitive polynomials $(t^9 + t^5 + 1, t^9 + t^4 + 1)$ by rotating and xoring. This gives two 9-bit correction values, which we concatenate to form a 3-byte FEC value which we send with Bob's flow (the ciphertext). This code is given in Listing 1.

```
typedef unsigned gf;
```

```
static gf mul_t_n(gf x, unsigned n, gf F) {
    // Return x*t^n mod [F = 0x221 or 0x211]
    for (; n>4; n-=4) x = (x<<4) ^ (x>>5) * F;
    return (x<<n) ^ (x>>(9-n)) * F;
}

gf melas_compute(const u8 *data, unsigned len) {
    gf r1=0, r2=0;
    for (unsigned i=0; i<len; i++) {
        r1 = mul_t_n(r1,8,0x221) ^ data[i];
        r2 = mul_t_n(r2,8,0x211) ^ data[i];
    }
    return r1 | r2<<9;
}
```

Listing 1: Code to compute Melas FEC

To decode, we calculate a syndrome $(s_1, \hat{s}_{-1})$ by xoring the given FEC with the calculated one, and let $s_{-1} :=$ bit-reverse$(\hat{s}_{-1})$. We will then work modulo only the polynomial $t^9 + t^5 + 1$. Let's assume for now that there are two errors at position e_1 and e_2, and let

$$E_1 := t^{e_1}, E_2 := t^{e_2}$$

We have

$$\begin{aligned} s_1 &= t^{e_1} + t^{e_2} = E_1 + E_2 \\ s_{-1} &= t^8 \cdot (t^{-e_1} + t^{-e_2}) = t^8/E_1 + t^8/E_2 \\ &= t^8 \cdot (E_1 + E_2)/(E_1 \cdot E_2) \end{aligned}$$

so that

$$s_2 := t^8 \cdot s_1/s_{-1} = E_1 \cdot E_2$$

Thus

$$Q(u) := u^2 + s_1 \cdot u + t^8 \cdot s_1/s_{-1} = 0$$

has roots at $u = E_1$ and $u = E_2$, and we may solve it easily using a half-trace computation. We may then repeatedly multiply E_1 and E_2 by t^8 modulo $t^9 + t^5 + 1$ until it has exactly one bit set; this bit is the error.

Conveniently, if there is only one error, a straightforward implementation of the above procedure gives $E_2 = 0$. Likewise if there are no errors, it gives $E_1 = E_2 = 0$. In either case, no modification is required to correct up to 2 errors.

The code to correct an error is listed in Figure 2. The authors are relatively inexperienced in error correcting codes, so this could probably be improved upon.

```
static gf mul(gf a, gf b) { // mod 0x221
    gf r = 0;
    for (unsigned i=0; i<9; i++) {
        r ^= (b>>i&1)*a;
        a = mul_t_n(a,1,0x221);
    }
    return r;
}

static gf reverse_bits_9(gf b) {
    b = (b&0x92) | (b>>2 & 0x49) | ((b&0x49)<<2);
    return (b&0x38) | (b>>6 & 0x7) | ((b&0x7)<<6);
}

void melas_correct(u8 *data, unsigned len, gf fec) {
    unsigned i,j;
    gf syndrome = fec ^ melas_compute(data,len);
    gf a = syndrome & 0x1FF, b = syndrome >> 9;
    gf r = mul(a,reverse_bits_9(b)), x=r, s=0;

    // Compute s = half_trace(t^8/x)
    for (i=0; i<7; i++) r = mul(mul(r,r),x);
    const u8 ht[9]={36,251,244,16,164,251,218,60,112};
    for (i=0; i<9; i++) s ^= ((r>>i)&1)*ht[i];

    a = mul_t_n(a,511-8*len,F1);
    s = mul(a,s<<1);

    for (j=0; j<2; j++, s^=a) {
        for (i=0, r=s; i<len; i++) {
            r = mul_t_n(r,8,F1);
            u32 mask = ((u32)(r & (r-1))-1)>>9;
            data[i] ^= r & mask;
        }
    }
}
```

Listing 2: Code to correct Melas FEC

Since we now only get a decryption error with at least 3 errors, this technique roughly cubes the probability of decryption failure [[**TODO: more precise**]]. We note that a stronger BCH code could correct more errors, but this would be slower and would take more work to implement. It may be worth using a larger code in the future to strengthen our parameters.

C Correctness

Let

$$\sum_{i=0}^{\deg(P)-1} e_i \cdot w^i := C_a := \text{clar} \cdot a^\top \cdot B$$

and

$$\sum_{i=0}^{\deg(P)-1} c_i \cdot w^i := C_b := \text{clar} \cdot b^\top \cdot A$$

Then if $e_i - c_i \in (-x/8, x/8) \bmod x$, the two parties will agree on a secret key. This is because modulo x, we have

$$\begin{aligned} c_i &= (K_b)_i \cdot x/2 + h_i \cdot x/4 + [0, x/4) \\ e_i &= d_i + h_i \cdot x/4 - x/8 + carry \\ &= (K_a)_i \cdot x/2 + [0, x/2) + h_i \cdot x/4 - x/8 + carry \end{aligned}$$

where $carry \in [-1, 1]$, so that

$$\begin{aligned} (e_i - c_i) &= ((K_a)_i - (K_b)_i) \cdot x/2 + [0, x/2) - [0, x/4) - x/8 + carry \\ &= ((K_a)_i - (K_b)_i) \cdot x/2 + [-3x/8, 3x/8] \end{aligned}$$

Therefore if $(K_a)_i \neq (K_b)_i$, we must have $|e_i - c_i| \geq x/8$ as claimed. Now,

$$E - C = a^\top \cdot (U(s)^\top b + \epsilon_b) - b^\top \cdot (U(s)a + \epsilon_a) = a^\top \epsilon_b - b^\top \epsilon_a$$

If the coefficients of this value are small enough, then decoding will be correct.

D Failure probability and chosen-ciphertext attacks

Here we quantify the failure probability for key exchange by explicitly computing the distribution of the difference of each coefficient of $E - C$. One way to do this is to rewrite the ring as

$$\mathbb{Z}[\phi, x]/(\phi^2 - \phi - 1, \phi - x^{\deg(P)/2})$$

We can then compute a distribution of coefficients in $\mathbb{Z}[\phi]/(\phi^2 - \phi - 1)$ and their products, and raise them to the appropriate powers to compute a distribution of $e_i - c_i$.

For decryption of public-key-encrypted messages, the failure model is more complicated for two reasons. First, there is the forward error correction to consider. We might expect our double-error-correcting code to cube the failure probability, but in fact there may be correlated failures if the ciphertext is particularly high-norm. Second, an attacker can search for such high-norm ciphertexts. Our implementation prevents the attacker from forming the ciphertext dishonestly, but the attacker can try different random seeds in order to maximize the probability of a failure.

To model this more complex scenario, we note that each coefficient of χ is in $\{-1, 0, 1\}$. However, multiplication can amplify this:

$$\begin{aligned}(a + b\phi) \cdot (c + d\phi) &= ac + bd + (ad + bc + bd)\phi \\ &= ac + bd + (ad + b(c + d))\phi\end{aligned}$$

Suppose $c + d\phi$ is noise in the ciphertext, and $a + b\phi$ is noise in the private key. Then the coefficients on a, b are in $\{0, \pm 1, \pm 2\}$, where ± 2 occurs only on b and only if $c = d = \pm 1$. Since the coefficients of the private key are independent, we model the ciphertext by the numbers $n_0, n_{\pm 1}, n_{\pm 2}$ of 0, ± 1 and ± 2 coefficients respectively, corresponding to a particular output position. For each triple $(n_0, n_{\pm 1}, n_{\pm 2})$ summing to $2 \cdot d \cdot \deg(P)$, we compute the distributions of the noise in the output and thus the probability

$$\epsilon_{i,n_{\pm 1},n_{\pm 2}} := \Pr(\text{error}|n_0, n_{\pm 1}, n_{\pm 2})$$

of a bit error. Likewise, for each triple we compute the probability

$$\delta_{i,n_{\pm 1},n_{\pm 2}} := \Pr(n_0, n_{\pm 1}, n_{\pm 2})$$

that the ciphertext will produce those coefficients. Note that this latter probability will be different for each output position.

We can then approximate, for a code that corrects e errors and an attacker who takes p^{-g} time to find a ciphertext that occurs with probability p (where $g = 1/2$ for Grover's algorithm), the normalized failure probability a given output position as:

$$F_{i,e,g} := \max_{p \in [0,1]} \sum_{\epsilon \geq p} \left(\epsilon_{i,n_{\pm 1},n_{\pm 2}}^{e+1} \cdot \delta_{i,n_{\pm 1},n_{\pm 2}} \right) \cdot \left(\sum_{\epsilon \geq p} \delta_{i,n_{\pm 1},n_{\pm 2}} \right)^{g-1}$$

Here the right-hand term is 1, and thus the max is at $p = 0$, in the classical case that $g = 1$ but not in the quantum case that $g = 1/2$.

Without error correction ($e = 0$) we can simply sum up $F_{i,e,g}$ across coefficients:

$$\text{effort}(e = 0, g) = \left(\sum_{i=0}^{\text{key length}} F_{i,e,g} \right)^{-1}$$

However, with error correction, computing separately for each coefficient will give the wrong answer unless we account for the relationship between $\Pr((n_0, n_{\pm 1}, n_{\pm 2})$ in position i) and in position j. But from the arithmetic-geometric means inequality it may be shown that

$$\text{effort}(e, g) \geq \left(\sum_{i=0}^{\text{key length}} F_{i,e,g} \right)^{-1} \cdot n^{-e}$$

[[**TODO: correlation at least between opposite coefficients**]]

There is a further possibility that ciphertexts may cause correlated failures that break error correction for reasons other than their norms, for example if they have regularly-spaced large coefficients. This may further reduce security against failure attacks. We leave analysis of this problem to future work.