Integer Module LWE key exchange and encryption: The three bears (draft)

Mike Hamburg*
might add Hart Montgomery and/or Arnab Roy†

September 12, 2017

Abstract

We propose a new post-quantum key exchange algorithm based on the integer module learning with errors (I-MLWE) problem. Our ThreeBears algorithm is simple and performant, but our main goal is to suggest MLWE over a generalized Mersenne field instead of a polynomial ring. We also show how to secure the system against chosen ciphertext attacks so that it can be used for public-key encryption.

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 [33]. Consequently, there is a growing interest in developing a suite of "post-quantum" algorithms which would resist attack by these computers [25]. The most common approaches to addressing this threat rely on the hardness of lattice problems, including variants such as learning

*Rambus Security Division
†Fujitsu research

with errors (LWE) [27], ring learning with errors (RLWE) [24], and module learning with errors (MLWE) [22].

Overall, RLWE-based schemes [17, 16, 2] tend to be faster and have smaller public keys and ciphertexts than those based on classical LWE [6]. This is enabled by the extra structure that the ring provides, but there is a lingering concern that this structure will enable new attacks [26]. This concern has led to proposals for rings with less structure [3], and for rings to be replaced with modules when the full ring structure is not necessary [7].

1.1 Our contribution

Here we propose a new cryptosystem based on *integer module learning with errors* (I-MLWE), where the underlying ring is the integers modulo a generalized Mersenne number. This opens up a new set of implementation options, and may have better (or worse!) security properties than a polynomial ring.

There are few systems based on lattices modulo generalized Mersenne numbers. Gu's concurrent work [12] covers some theoretical ground but contains no concrete proposals, and Joux et al.'s system [1] offers less security than intended [5]. We aimed to produce a practical I-MLWE system as a starting point for analysis. To do this, we modified Kyber [7] to use I-MLWE instead of MLWE. We call this cryptosystem ThreeBears, because its modulus has the same shape as the one in Ed448-Goldilocks [15].

We also investigated possible changes beyond I-MLWE. When choosing parameters for LWE encryption algorithms, there is a tradeoff: more noise means higher security but also a higher risk of decryption failure. For key exchange, a small failure rate is mostly harmless, but for encryption it can lead to lost data, or worse, a chosen-ciphertext attack [18]. Saarinen proposed to improve this tradeoff by using an error-correcting code [30, 29]. We include a thorough analysis of this technique. Our analysis is mostly unrelated to I-MLWE and may be of independent interest.

Our system thus draws inspiration from several sources: Lyubashevsky-Peikert-Regev [23] and Ding [20]'s LWE protocols; KYBER's overall structure [7]; Saarinen's error correction [30, 29]; and generalized Mersenne ring structure with Goldilocks' prime shape [15]. The resulting system is competitive with other state-of-the-art RLWE KEMs in terms of simplicity, performance and conjectured security level.

1.2 Notation

Let $\mathbb{Z}$ denote the integers. Let $\{0,1\}^n$ denote the set of n-bit strings, and let $\mathcal{B}^n$ denote the space of n-byte strings. For a ring R, let R^d and $R^{d\times e}$ be the spaces of d-dimensional vectors and $(d \times e)$-dimensional matrices over R, respectively.

2 Generalized Mersenne rings

2.1 Polynomial rings

Most previous systems use polynomial rings of the form

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

for some integer q and polynomial P of degree D. They also specify one or more noise distributions χ_i over R_{poly}, typically

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

where ψ is a distribution on the integers — perhaps a binomial [2], uniform or discrete Gaussian [8] distribution. Alternatively, the noise may be chosen to have a fixed Hamming weight [3], or may be a consequence of rounding [11]. 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 forces the polynomial P to be sparse with small coefficients.

2.2 Rings modulo generalized Mersenne numbers

Here we take a slightly different approach. We choose an integer x and a polynomial P of degree D, 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 generalized Mersenne (GM) number N. The integer x plays the role of both the modulus q and the formal variable x in a polynomial ring. We generate the noise in the analogous way:

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

Again, we need P to be sparse with small coefficients.

2.3 Pros and cons of pseudo-Mersenne rings

Both polynomial rings and GM rings have their advantages. Here is a brief comparison.

Security Polynomial rings have been studied for longer, which makes them a more conservative choice. Gu [12] shows a reduction between LWE problems on polynomial and GM rings. The reduction isn't tight: it increases the noise by a factor of D in either direction. So it is meaningless for practical systems, but it suggests that RLWE and I-RLWE may have similar security.

Speed Cyclotomic polynomial rings that split mod q support very fast in-place multiplication using the number-theoretic transform (NTT). These rings use small coefficients, so they work well on small machines and they vectorize well. GM rings don't have these advantages, but they can take advantage of a large multiplier and any existing hardware or software support for multi-precision arithmetic. Overall, their speed is competitive with polynomial rings.

Simplicity GM rings have complexity due to carry propagation. But it's easy to convert elements to and from bytes without wasting space, and it's easy to sample uniformly from the ring. There is also no need to send or store anything in the NTT domain.

Noise amplification Non-cyclotomic rings amplify noise when reducing modulo P or N, which leads to a slightly worse security vs. space trade-off. GM rings can also be cyclotomic [12], but at a cost in complexity and performance. Our ring amplifies noise by 3/2, which is the best for the non-cyclotomic shapes we studied.

Summary Overall, we believe that generalized Mersenne rings are about as suitable for this task as polynomial rings. These rings have received less attention for lattice problems, and we are proposing ThreeBears as a step toward correcting this gap.

2.4 Module-LWE

It is most convenient to exchange keys whose bit length is less than or equal to the dimension of the ring. Post-quantum systems must contend with Grover's algorithm [14], so a 256-bit key is appropriate for high-security systems.

We have chosen a ring of dimension 312, which allows us to transport a 256-bit key with error correction. However, this is too small a dimension to effectively resist lattice reduction algorithms such as BKZ [32, 10]: a dimension around 500-1000 is more appropriate. Like Kyber [7], we address this problem by using the vector space R^d for some small dimension d.

2.5 Noise distribution

We will make our noise distribution on the ring by applying a distribution ψ_{σ^2} to each coordinate, where σ^2 is the variance. When $\sigma^2 \leq 1/2$, 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 $\sigma^2 > 1/2$, let $\psi_{\sigma^2} := \psi_{1/2} + \psi_{\sigma^2-1/2}$. Over the ring, we will sample ψ_{σ^2} independently for each coefficient. Let

$$\chi_{\sigma^2} := \sum_{i=0}^{D-1} \epsilon_i \cdot x^i \in R \quad \text{where} \quad \epsilon_i \leftarrow \psi_{\sigma^2} \text{ independently}$$

Over the module, let $\chi_{\sigma^2}^d$ sample from R^d by sampling each coordinate i.i.d. from χ_{σ^2}.

2.6 Sampling with coins

Let $\mathcal{D}$ be some distribution, or a set in which case we mean the uniform distribution on that set. Let $\text{samp}(\mathcal{D},\ c,\ n)$ be an efficient deterministic algorithm which samples $\mathcal{D}$ using short random coins c and optional nonce n. In our initial implementation, we do this by computing a stream of pseudorandom values using the extendible-output function [[**TODO: something like**]]

$$\text{cSHAKE256}(\text{“ThreeBears v0.1”}, c||n)$$

and then converts that stream into a sample of $\mathcal{D}$.

[[**TODO: distinguish ROM from PRF?**]]

2.7 Clarifier

For non-cyclotomic rings, multiplying two samples of the noise distribution produces a larger combined noise than necessary. We mitigate this by applying a *clarifier*. Let $\text{clar} \in R^*$ be chosen to minimize the variance of the

coefficients of

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

If $P = x^D - x^{D/2} - 1$, then the optimal clarifier is

$$\text{clar} = x^{-D/2} = x^{D/2} - 1$$

Because this clarifier matches the ring structure, multiplying by it is essentially free.

2.8 Recommended parameters

For our main recommendations, we use the "golden Solinas" prime

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

with $x = 2^{10}$ and $P = x^{312} - x^{156} - 1$. We analyze other parameter choices in Appendix A, including different rings and toy parameters.

Our recommendations are given in Table 1. They all transport 256-bit keys, so they have at most 128 bits of security against quantum attacks. They have differing security margins against lattice attacks and chosen-ciphertext attacks. We believe that the lattice attacks have more room for improvement, and so have tuned the parameters to have a higher security margin against them. The "+" instances use roughly 50% more noise, but keep a low failure probability by applying 2-bit forward error correction. Our main recommendation is MamaBear.

3 CPA-secure key encapsulation

GM rings are suitable for many of the same protocols that polynomial rings are used for. We begin by describing key encapsulation mechanism (KEM) along the lines of Lyubashevsky-Peikert-Regev [23] or Ding [20]. This is shown in Figure 1. For this KEM, Alice and Bob's secrets are ephemeral and must never be reused, because otherwise a chosen-ciphertext attack

System	$d \cdot D$	x	σ^2	FEC	Failure	PQ Security Lattice	PQ Security CCA
BabyBear	$2 \cdot 312$	2^{10}	$3/8$	0	2^{-133}	130	120
MamaBear	$3 \cdot 312$	2^{10}	$9/32$	0	2^{-153}	201	138
PapaBear	$4 \cdot 312$	2^{10}	$1/4$	0	2^{-148}	275	134
BabyBear+	$2 \cdot 312$	2^{10}	$19/32$	2	2^{-148}	140	134
MamaBear+	$3 \cdot 312$	2^{10}	$1/2$	2	2^{-147}	218	137
PapaBear+	$4 \cdot 312$	2^{10}	$3/8$	2	2^{-188}	290	173

Table 1: Main recommendations, failure probabilities, and log security estimates against quantum attacks. [[**TODO: Validate vs DropBear, consider** $7/16$ **for PapaBear+, tighten failure computations, and rebuild this table and the comparison one**]]

might recover the private keys and messages. The mechanism consists of three algorithms: **KeygenCPA**, **EncapsCPA** and **DecapsCPA**.

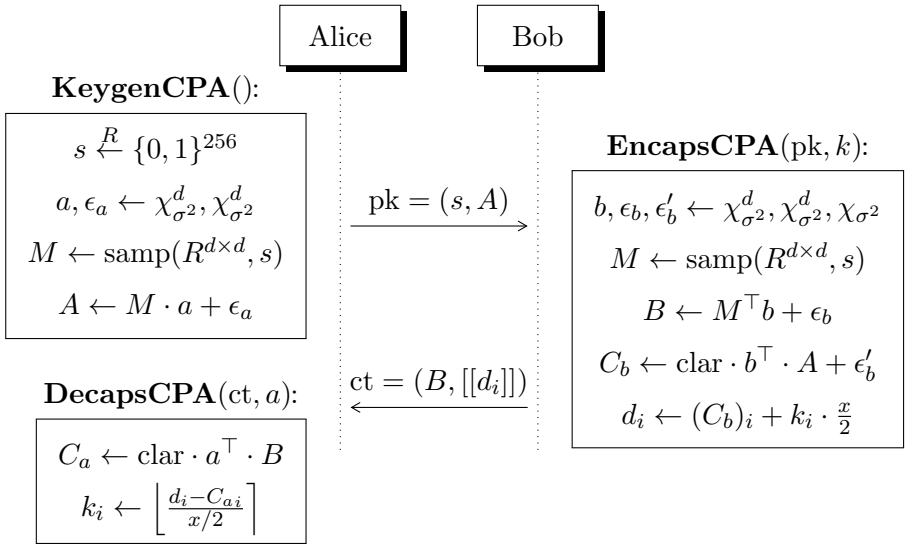

Figure 1: CPA-secure key exchange sketch. In the actual protocol, only $\ell = 4$ bits of each d_i are sent.

3.1 KeygenCPA

The key exchange mechanism begins by generating an ephemeral key. The key must never be reused; see Section 4 for a version which supports key reuse.

KeygenCPA proceeds as follows:

$$\begin{aligned} s &\xleftarrow{R} \mathcal{B}^{\text{matrix\_seed\_len}} \\ a,\ \epsilon_a &\xleftarrow{R} \chi^d_{\sigma^2},\ \chi^d_{\sigma^2} \\ M &\leftarrow \text{samp}(R^{d\times d},\ s) \\ A &\leftarrow M \cdot a + \epsilon_a \end{aligned}$$

The public key is (s, A) and the private key is a. In practice, s, a and ϵ_a will also be sampled from coins, so a space-constrained implementation could simply store those coins. In that case, the initial coins should be at least 320 bits long to prevent multi-target attacks.

The matrix_seed_len parameter is 32 bytes for all recommended instances.

3.2 EncapsCPA

The **EncapsCPA**$((s, A);\ k)$ algorithm encrypts an n-bit symmetric key k using the public key (s, A). The length n of key can be up to most the dimension D. **EncapsCPA** first chooses secrets

$$b,\ \epsilon_b,\ \epsilon'_b \leftarrow \chi^d_{\sigma^2},\ \chi^d_{\sigma^2},\ \chi_{\sigma^2}$$

and computes

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

Now we will use C_b to encapsulate a key k, using an approach similar to LPR [23], Ding [20] and KYBER [7]. For LPR, we would output $C_b +$ Encode(k) for a suitable encoding function. But this wastes bits by sending

more coefficients than necessary (d instead of n), and more bits per coefficient. Ding's approach sends only the second-highest bit of each coefficient (or rather, the highest coefficient after doubling), and extracts the highest bit as the shared key. But we will need to choose the key for CCA-secure key encapsulation. We also found that Ding's approach sends too little information per coefficient (1 bit, and we would send a 2nd in order to encrypt the key). We found that the ratio of estimated log security to bandwidth is maximized by sending 3-4 bits per coefficient, like KYBER. We chose 4 for implementation simplicity reasons.

Specifically, to encrypt the ith key bit, we extract ℓ bits of the e_ith coefficient, where the parameter $\ell = 4$ for all recommended instances and the position

$$e_i := \begin{cases} i/2 & \text{if } i \text{ is even} \\ D - (i+1)/2 & \text{if } i \text{ is odd} \end{cases}$$

is the one with the ith-least noise amplification. Call these extracted bits

$$\text{extract}(\ell, i, C_b) := \left\lfloor \frac{(C_b)_i}{x/2^\ell} \right\rfloor \text{ where } C_b = \sum_{i=0}^{D-1} C_{b,i} \cdot x^i \text{ and } 0 \le C_{b,i} < x$$

The encryption part of the capsule is then

$$d_i := \text{extract}(\ell, i, C_b) + k_i \cdot 2^{\ell-1} \bmod 2^\ell$$

The output of **EncapsCPA** is the capsule

$$(B,\ D) \text{ where } D := [[d_i]]_{i=0}^{n-1}$$

To encrypt an arbitrary-length message, we suggest a KEM-DEM approach as in [13].

The addition of ϵ'_b is probably unnecessary, but if it is omitted then our security is based on learning with rounding instead of the better-studied learning with errors.

3.3 DecapsCPA

The decapsulation algorithm **DecapsCPA**$(a, (B, D))$ decrypts the encapsulated key from the capsule and private key. It computes

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

It then subtracts this from the encoded key and finds the result by rounding:

$$k_i := \left\lfloor \frac{2 \cdot d_i - \text{extract}(\ell + 1, i, C_a)}{2^\ell} \right\rceil$$

Note that d_i is doubled only because we extract one more bit on decapsulation. The output of **DecapsCPA** is the key $k := [[k_i]]_{i=0}^{n-1}$.

The resulting key is almost always the same as the k that was input to **EncapsCPA**, but with some small probability the key exchange will fail. See Appendix B for an analysis of the failure probability.

3.4 Forward error correction

For any LWE system, adding more noise increases the security but also the failure probability. Saarinen suggested using forward error correction (FEC) to decrease the failure probability, allowing for higher noise [30, 29]. We adopted this design for ThreeBears.

Failures consist of mostly uncorrelated single-bit errors, so we needed a code that corrects bit errors. We implemented a Melas-style BCH$(511, 493, 5)$ code [21], which can correct up to 2 errors in up to 511 bits at the cost of 18 bits of overhead. So in addition to a 256-bit key, we can encrypt 18 bits of forward error correction for a total of 274 bits. This gives "+" versions of our recommended parameters, which add $5-8\%$ extra security, and an extra margin against hybrid attacks, at a cost of 9 extra bytes of ciphertext.

Since we could encrypt up to $D = 312 > 256 + 9 \cdot 6$ bits, we could correct up to 6 errors with a BCH code, but we found it very complex to correct more than 2 errors in constant time.

We analyzed the effect of forward error correction on failure probability for both key exchange and encryption. See Appendix B for the analysis, and Appendix B.6 for a discussion of just how much this tradeoff buys us.

For CCA security the failure probability needs to be cryptographically negligible, but for ephemeral key exchange it doesn't. Without error correction, the noise parameters of BabyBear+, MamaBear+ and PapaBear+ give failure probabilities of about 2^{-59}, 2^{-57} and 2^{-71} respectively, which is perfectly acceptable for key exchange.

4 CCA-secure Key encapsulation

In the (quantum) random oracle model, we can convert this ephemeral key exchange into a CCA-secure key exchange (and thus a public-key encryption algorithm) with a variant of the Targhi-Unruh conversion [34]. This will define algorithms **KeygenCCA**, **EncapsCCA** and **DecapsCCA**.

These algorithms use hash functions, just like the CPA versions. We require those hash functions to be appropriately domain-separated, so that a CPA key or ciphertext can't be used as a CCA one, or vice versa. [[**TODO: implement**]]

4.1 KeygenCCA

The key generation algorithm is the same, except that the private key is (a, pk_a) instead of just a. That is, the public key is needed for decryption.

4.2 EncapsCCA

Let **EncapsCPA**$(s, A;\ k;\ c)$ be the CPA-secure encapsulation algorithm with an addition input of coins c. Instead of sampling from χ_{σ^2} or ψ_{σ^2}

at random, that algorithm samples them using $\text{samp}(\chi_{\sigma^2};\ c;\ i)$ for the ith sample.

We will use this to build a CCA-secure encapsulation algorithm using a variant of the Targhi-Unruh transform [34]. As a consequence of this transform, the sender doesn't get to choose the encapsulated key, so instead it is an output of **EncapsCCA**.

The function $\textbf{EncapsCCA}(s, A)$ works as follows:

$$\begin{aligned} k_0 &\xleftarrow{R} \{0,1\}^{\text{seed\_length}} \\ \text{iv} &\xleftarrow{R} \{0,1\}^{\text{iv\_length}} \\ c, \delta, k_1 &\leftarrow H(s||A||k_0) \\ C, D &\leftarrow \textbf{EncapsCPA}(s, A;\ k_0;\ \text{iv}||c) \end{aligned}$$

The capsule is then $(\text{iv}, C, D, \delta)$ and the shared secret is k_1.

The recommended parameters are as follows:

- Seed length: 256 bits. This gives 256-bit resistance to brute force attacks by classical adversaries and at least 128-bit resistance to quantum ones.
- IV length: 0 bits. The IV is useful for preventing classical multi-target attacks on many messages encrypted with the same public key. Such attacks take 2^{192} hash operations, and so are impractical. However, to reach NIST's security category 5 against multi-target attacks, implementors could set this to 64 bits.
- Length of δ: 0 bits. Extra-conservative users may want the reassurance that the Targhi-Unruh δ values provides, but we are confident that it adds no security. See Appendix C for details.
- Length of c: 320 bits, to prevent multi-target attacks.

4.3 DecapsCCA

The **DecapsCCA** algorithm recovers a shared secret from the private key (a, pk_a) and a capsule $(\mathrm{iv}, C, D, \delta)$.

It does this by recovering a

$$k_0 \leftarrow \mathbf{DecapsCCA}(C, D)$$

It then runs

$$(\mathrm{capsule}', k_1) \leftarrow \mathbf{EncapsCCA}(\mathrm{pk}_a; \mathrm{iv}, k_0)$$

If capsule$'$ is the same as the received capsule, then decapsulation is successful and the shared secret is k_1. Otherwise decapsulation fails.

The Targhi-Unruh tag δ isn't used in decapsulation, except that re-encapsulation must produce the same δ.

4.4 Security analysis

[[TODO: Need a security proof]]

There are three clear avenues of attack against ThreeBears. The first is to brute-force the seeds or transported keys using Grover's algorithm [14]. Those keys are all 256 bits, so this takes about 2^{128} effort.

The second avenue is to attack the I-MLWE problem itself, most likely with a lattice attack such as BKZ [32, 10]. We estimated the "core quantum SVP hardness" of this attack using NewHope's BKZ parameter estimation scripts. We also used John Schanck's estimator [31] to estimate the cost of a hybrid attack [9]; in all cases, this was estimated to be harder than a direct attack. These estimates should be very conservative, but we wanted a large security margin because lattice attacks have the most room for improvement.

The third avenue is a chosen-ciphertext attack on the supposedly-CCA-secure KEM, where the attacker would gain information by causing decryption failures. The attacker could even use a quantum computer to find

chosen ciphertexts that are more likely to fail. We analyze this attack in detail in Appendix B.5. Our analysis uses conservative approximations, but it is not provably tight so we have left a security margin. In addition to being computationally infeasible, this attack would require an impractical number of chosen ciphertexts.

	PQ Security			Size in bytes		
System	SVP	CCA	Failure	SK	PK	CT
BabyBear	130	120	133	780	812	908
MamaBear	201	138	153	1170	1202	1298
PapaBear	275	134	148	1360	1592	1688
BabyBear+	140	134	148	780	812	917
MamaBear+	218	137	147	1170	1202	1307
PapaBear+	290	173	188	1360	1592	1697
Kyber light [7]	102	169^c	169	832	736	832
Kyber rec. [7]	161	142^c	142	1248	1088	1184
Kyber paranoid [7]	218	145^c	145	1664	1440	1536
JarJar [2]	118	-	55	896	928	1024
NewHope [2]	255	-	61	1792	1824	2048
trunc8 [30]	131	-	45	128	1024	1024
Hila5 [29]	255	135^c	135	1792	1824	2012
NTRU ees743ep1 [17]	159	-	112	1120	1022	1022
S.NTRU′ 4591^{761} [3]	126	∞	∞	382	1218	1047
NTRU KEM [19]	123	∞	∞	382	1140	1281

Table 2: Security and message sizes for ThreeBears and related work. Security estimates are log base 2 of the conservatively estimated attack effort. Failure is -log base 2 of the failure probability. Entries marked ^{c} assume a classical adversary. Private key sizes don't include the public key.

[[TODO: Reconcile our numbers for NTRU Prime, and DJB's and Kyber's]]

We compare the security of ThreeBears to similar systems in Table 2. The biggest cost to increasing the security parameters for a Ring-LWE system

is that the public key and messages become larger, so we compare this information as well.

[[TODO: multi-target attacks. Probably can't get private key (320-bit seed), maybe can get ciphertexts if all encrypted to same key, maybe need to salt everything.]]

5 Performance

We created a an implementation of ThreeBears in C, which is optimized for performance, but also for simplicity and memory consumption. It uses two levels of Karatsuba multiplication, and can take advantage of the Keccak Code Package's SIMD implementations (KCP) [4]. On x86-64, the code uses a multiply-accumulate intrinsic in the inner loop, which gives a 10% performance boost on Skylake. On other processors, it contains no assembly language, but it can take advantage of $64 \times 64 \to 128$-bit multiplication if the CPU and compiler support `__uint128_t`.

Our implementation follows this paper's description of the private key as the vector a in serialized form, plus the public key for CCA modes. There are several possible size-speed tradeoffs. We could compress a at little cost, because its coefficients are in $[-1, 1]$[1], or we could skip serializing it. We could store only a seed and re-derive the public and private key. Or we could cache $\text{samp}(R^{d \times d},\ s)$, which would speed up re-encryption. We didn't benchmark these tradeoffs.

We benchmarked our code on several different platforms. The results are shown for Intel Skylake in Table 3; for ARM Cortex-A53 in Table 4; and for ARM Cortex-A8 in Table 5. These are intended to represent computers, smartphones, and embedded devices respectively **[[TODO: tiny IOT m3 or AVR]]**. Each table shows the compilation options used. We compared ThreeBears's performance to NewHope and Kyber. We see that

[1] Except for BabyBear+, where they're in $[-2, 2]$.

out primary recommendation, MamaBear, is competitive with Kyber and NewHope on Skylake.

The comparison isn't fair on the ARM systems [[**TODO: So remove it? Or bench against Curve25519?**]], because Kyber and NewHope have more to gain from machine-specific optimizations. On the Cortex-A8, both systems would benefit handily from NEON vectorization, but ThreeBears links against KCP and its NEON-optimized Keccak code. On Cortex-A53, ThreeBears doesn't have this advantage (Keccak is faster in the scalar unit) but it probably won't gain anything from vectorization and NewHope/Kyber probably will. Still, the comparison shows that ThreeBears's performance is at least respectable on those platforms.

For brevity, we omit benchmarks of the "+" versions with forward error correction, which costs only about 1% extra runtime. BabyBear+ is another $\sim 5\%$ slower because the sampler does extra work when $\sigma^2 > 1/2$.

6 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 Conclusion

In this paper, we presented ThreeBears, a relatively simple instantiation of module LWE based on generalized Mersenne numbers. This system provides an alternative to polynomial rings for ring- and module-LWE instances.

		Ephemeral		CCA-secure	
System	Keygen	Encaps	Decaps	Encaps	Decaps
BabyBear	40k	56k	13k	64k	77k
MamaBear	83k	97k	17k	108k	126k
PapaBear	123k	146k	22k	160k	183k
NewHope AVX2	88k	118k	18k	-	-
Kyber AVX2	75k	-	-	110k	114k

Table 3: Performance in cycles on a NUC with Intel Core i3-6100U "Skylake" 64-bit processor at 2.3GHz. Compiled with `clang-3.9 -O2 -DNDEBUG -march=native`

		Ephemeral		CCA-secure	
System	Keygen	Encaps	Decaps	Encaps	Decaps
BabyBear	153k	205k	47k	217k	265k
MamaBear	302k	370k	65k	389k	456k
PapaBear	498k	584k	84k	607k	692k
NewHope ref	589k	913k	236k	-	-
Kyber ref	550k	-	-	751k	921k

Table 4: Performance in cycles on a Raspberry Pi 3 with Cortex-A53 64-bit processor at 1.2GHz. Compiled with `clang-3.9 -O2 -DNDEBUG -mcpu=cortex-a53`

		Ephemeral		CCA-secure	
System	Keygen	Encaps	Decaps	Encaps	Decaps
BabyBear	363k	510k	139k	532k	678k
MamaBear	767k	970k	202k	1008k	1217k
PapaBear	1303k	1565k	266k	1605k	1886k
NewHope ref	1026k	1552k	377k	-	-
Kyber ref	1514k	-	-	1939k	2152k

Table 5: Performance cycles on a BeagleBone Black with Cortex-A8 32-bit processor at 1GHz. Compiled with `clang-3.9 -O2 -DNDEBUG -march=native`

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. We have shown that generalized Mersenne module-LWE performs competitively with other module-LWE key exchange mechanisms.

We also improved the analysis of error correcting codes to reduce the failure probability of module-LWE key exchange. Our techniques for that problem may be of independent interest.

7.1 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 [25]. 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.

7.2 Acknowledgements

Special thanks to John Schanck for his help in analyzing the hybrid attack.

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] 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.

[4] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Vladimir Sedach. Keccak code package, 2017. https://github.com/gvanas/KeccakCodePackage.

[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 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.

[8] 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.

[9] Johannes Buchmann, Florian Göpfert, Rachel Player, and Thomas Wunderer. On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack. In *Africacrypt 2016*, 2016. http://eprint.iacr.org/2016/089.

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

[11] 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`.

[12] Gu Chunsheng. Integer version of ring-LWE and its applications. Cryptology ePrint Archive, Report 2017/641, 2017. `http://eprint.iacr.org/2017/641`.

[13] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[14] 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.

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

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

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

[18] 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.

[19] Andreas Hülsing, Joost Rijneveld, John Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. CHES, 2017. http://eprint.iacr.org/2017/667.

[20] 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.

[21] Gilles Lachaud and Jacques Wolfmann. Sommes de kloosterman, courbes elliptiques et codes cycliques en caractéristique 2. *CR Acad. Sci. Paris Sér. I Math*, 305(20):881–883, 1987.

[22] 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.

[23] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010*, 2010. http://eprint.iacr.org/2012/230.

[24] 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.

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

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

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

[28] Miruna Rosca, Amin Sakzad, Ron Steinfeld, and Damien Stehle. Middle-product learning with errors. *CRYPTO 2017*, 2017. http://eprint.iacr.org/2017/628.

[29] 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.

[30] 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.

[31] John Schanck. LWE hybrid attack scripts. Personal communication.

[32] 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.

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

[34] 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.

[35] Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. *International Journal of Quantum Information*, 13(04):1550014, 2015.

A Other ring choices

Readers may be curious why we chose this specific ring

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

Certainly some sort of generalized Mersenne number is required to minimize noise amplification, but why this one? 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 $2^{12\cdot 268+1}-1$, which means that clar $= 2$ would work nicely. However, the noise 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}.

It is not obvious that the modulus must even be prime. It seems likely that sparse factors would lead to attacks, but possibly a generalized Mersenne or Fermat number could be secure if it had no sparse factors. We didn't want to gamble on this.

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 noise 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, $\Phi_{2\cdot 607\cdot 13}(2)$ works with clarifier $2^{280\cdot 13+1} - 2^{140\cdot 13} - 1$. We did not see an appreciable gain in this approach, but at least it's mathematically interesting.

Middle product learning with errors [28] would probably work with integers, at some cost in performance.

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

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

i.e. a polynomial ring over a GM field. We didn't do this because we wanted to keep things simple.

Within GM primes, golden-ratio ones seem to provide the smallest noise amplification and the widest selection of implementation choices. Within golden-ratio primes, our choice was driven by need for a digit size of at least 2^{10} with a degree at least 256. It can also be used with $x = 2^{12}$ and $D = 260$, but our estimates suggest that this has worse security. This might be useful if hybrid attacks improve. The prime $2^{2600} + 2^{1300} - 1$ is also a good option,

System	$d \cdot D$	σ^2	x	FEC	Security Hybrid	Security Failure
GummyBear	$1 \cdot 270$	$7/32$	2^8	0	50	42
TeddyBear+	$1 \cdot 390$	$7/32$	2^8	2	81	80
DropBear	$2 \cdot 312$	$5/4$	2^{10}	0	157	12

Table 6: Toy bears.

but it's more annoying to implement (due to being $2^n + \epsilon$ instead of $2^n - \epsilon$) and it doesn't match the NIST security levels quite as well.

We chose a small ring size to make it easier to choose parameters by varying d. If we wanted a ring-LWE solution (rather than module-LWE) with a large modulus, we could use $2^{2 \cdot 12 \cdot 905} - 2^{12 \cdot 905} - 1$ or the "plastic" prime $2^{12 \cdot 720} - 2^{12 \cdot 480} - 1$.

A.1 Toy parameters

We propose toy parameters to encourage cryptanalysis. Our first two toy parameter sets, GummyBear and TeddyBear, have smaller digits, smaller degrees and less noise. The noise is small enough to expose them to hybrid lattice/meet-in-the-middle attacks [9]. Our third toy, DropBear, is like BabyBear but with twice the noise and no error correction, which exposes it to failure attacks.

A.2 Other noise distributions

Our ring and noise distribution result in uneven noise amplification: coefficients near $x^{D/2}$ in the final result have more noise, and those near x^0 and x^D have less. It might be worth shaping the noise distribution to counter this problem. For example, we could add less noise to some coefficients, or we could add correlated noise. This didn't seem to be worth the extra complexity of analysis and implementation.

Another option would be to use noise with a fixed Hamming weight, like NTRU Prime. This would lower the failure rate, but we decided that independent noise would be easier to implement.

B Failure probability and chosen-ciphertext attacks

B.1 Correctness criterion

Let

$$\sum_{i=0}^{D-1} a_i \cdot w^i := C_a := \text{clar} \cdot a^\top \cdot B$$

and

$$\sum_{i=0}^{D-1} b_i \cdot w^i := C_b := \text{clar} \cdot b^\top \cdot A + \epsilon'_b$$

and suppose we are extracting ℓ bits for encryption. Then if

$$|(a_i - b_i) \bmod x| \leq (1 - 2^{1-\ell}) \cdot \frac{x}{4}$$

the decryption will be successful. This is because modulo x, we have

$$\text{extract}(C_b, i, \ell) = \left\lfloor \frac{b_i}{x/2^\ell} \right\rceil = \frac{b_i}{x/2^\ell} - \frac{1}{2} - t_{b,i} \text{ where } t_{b,i} \in \left[-\frac{1}{2}, \frac{1}{2} - \frac{2^\ell}{x} \right]$$

The sender sends

$$d_i \quad = \quad \frac{b_i}{x/2^\ell} - \frac{1}{2} - t_{b,i} + k \cdot 2^{\ell-1}$$

and the recipient extract the key bit

$$\begin{aligned} k'_i &= \left\lfloor \frac{2 \cdot d_i - \text{extract}(\ell+1, i, C_a)}{2^\ell} \right\rceil \\ &= \left\lfloor \frac{2 \cdot d_i - \left\lfloor \frac{a_i}{x/2^{\ell+1}} \right\rceil}{2^\ell} \right\rceil \\ &= \left\lfloor \frac{2 \cdot d_i + 1 - \left\lceil \frac{a_i+1}{x/2^{\ell+1}} \right\rceil}{2^\ell} \right\rceil \end{aligned}$$

Here the numerator is an integer, so the inner ceiling doesn't change the rounding. We now have

$$\begin{aligned} k_i' &= \left\lfloor \frac{2 \cdot d_i + 1 - \frac{a_i+1}{x/2^{\ell+1}}}{2^\ell} \right\rceil \\ &= \left\lfloor \frac{2 \cdot (b_i - a_i - 1)}{x} + \frac{2 \cdot k \cdot 2^{\ell-1} - 2 \cdot t_{b,i}}{2^\ell} \right\rceil \\ &= k + \left\lfloor \frac{2 \cdot (b_i - a_i)}{x} - \frac{1}{x} - \frac{t_{b,i}}{2^{\ell-1}} \right\rceil \end{aligned}$$

Then by assumption

$$\frac{|2 \cdot (a_i - b_i)|}{x} \leq \frac{1}{2} - \frac{1}{2^\ell}$$

and

$$\left| \frac{t_{b,i}}{2^{\ell-1}} + \frac{1}{x} \right| \leq \frac{1}{2^\ell}$$

so the rounded quantity is less than $1/2$ and rounds to 0.

B.2 Failure probability without error correction

Here we quantify the failure probability for key exchange by computing the distribution of $a_i - b_i$. One way to do this is to rewrite the ring as

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

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 $a_i - b_i$.

However, there are two additional wrinkles. 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 (e.g. if the ciphertext is particularly high-norm). Second, an attacker can search for such failure-prone 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 $c = d = \pm 1$.[2]

Since the coefficients affect the variance of the ciphertext, a coefficient of ± 2 is roughly four times worse than a coefficient of ± 1. We performed some of the analyses in this section twice, in one case tracking the number of ± 1 and ± 2 coefficients, and in the other case tracking the variance. The results were very similar, and tracking only the variance was much faster, so we adopted that approach for all our analyses.

Let r be a ring element; it may be written in signed notation[3] as

$$r = \sum_{i=0}^{D-1} c_i x^i \quad \text{where} \quad c_i \in [-x/2, x/2)$$

Define its norm as

$$\text{norm}(r) := \sum_{i=0}^{D-1} c_i^2$$

We will define the norm of a ciphertext $U^\top b + \epsilon_b$ with respect to output position i as

$$\text{norm}_i(b, \epsilon_b) := \sum_{j=0}^{d-1} \big(\text{norm}(b \cdot \text{clar} \cdot x^i) + \text{norm}(\epsilon_b \cdot \text{clar} \cdot x^i)\big)$$

For each possible norm n, we computed the distributions of the noise in the output, plus additional noise of variance $< 1 + \sigma^2$ due to carries and the

[2]The situation is more complicated for BabyBear+ since it has a larger noise, but the method applies.

[3]This representation is unique except for elements with huge norm.

addition of ϵ'_b. With our rounding mechanism, the error probability ramps from 0 at $x/8$ to 1 at $3x/8$, and the conditional probability of a single bit error is approximately

$$\begin{aligned}
\epsilon_n &:= \Pr(\text{bit error} \mid n) \\
&\leq \sum_{z=x/8}^{n} \frac{z - x/8}{x/4} \cdot \Pr((\text{output difference is } \pm z) \mid n) \\
&= \sum_{z=x/8}^{n} \left(\frac{z - x/8}{x/4} \cdot \sum_{\substack{k=z \\ k+z \text{ even}}}^{n} \binom{n}{k} \binom{k}{(k+z)/2} \frac{\sigma^{2k} \cdot (1-\sigma^2)^{n-k}}{2^{k-1}} \right)
\end{aligned}$$

Likewise, for each norm n and position i we computed the probability

$$\delta_{i,n} := \Pr(\text{norm}_i(\text{ciphertext}) \text{ is } n)$$

that a random ciphertext will have that norm in position i. We did this by convolving the distributions that each pair of opposite coefficients of the ciphertext contributes to the norm.

After extracting $B := 256$ bits, the failure probability without error correction is then at most

$$p_0 := \Pr(\text{bit failure anywhere}) = \sum_{i=0}^{B-1} \sum_{w=0}^{5d \cdot D} \epsilon_w \cdot \delta_{i,w}$$

by the union bound.

B.3 With error correction

When error correction is used, the calculation becomes more complex. We might hope that the probability of an error after e errors have been corrected would be p_1^{e+1}, but that would require assuming that failures are uncorrelated. Unfortunately they are correlated in multiple ways, which we have not fully studied. We have pinpointed and evaluated three causes of correlation, which we believe to be the three most important:

- Ciphertext norm: the larger the norm of the ciphertext, the higher the probability of failaure.
- Secret key norm: the larger the norm of the secret key, the higher the probability of failure.
- Correlation: some output bits are correlated for all ciphertexts.

We did not study the separate problem that some output bits may be correlated for a *particular* ciphertext, for example if the ciphertext has many regularly spaced coefficients.

As for ciphertext norm, we already have the right tool $\delta_{i,n}$ to take care of that. For secret key norm, we can define a corresponding γ_m which is the probability that the secret key has norm m, and change ϵ_w to $\epsilon_{m,n}$ which is the probability of an error with secret key of norm m and ciphertext of norm n.

After correcting up to e errors in $B := 256 + 18$ bits, with a ciphertext ct, we would expect a total error probability of

$$\begin{aligned} p_e(\mathrm{ct}) &= \sum_{\text{key } k} \mathrm{pr}(k) \cdot \sum_{|E|=e+1} \prod_{i \in E} \epsilon_{\mathrm{norm}_i(k),\mathrm{norm}_i(\mathrm{ct})} \\ &< \frac{1}{(e+1)!} \cdot \sum_{\text{key } k} \mathrm{pr}(k) \cdot \left(\sum_{i=0}^{B-1} \epsilon_{\mathrm{norm}_i(k),\mathrm{norm}_i(\mathrm{ct})} \right)^{e+1} \\ &\leq \frac{B^e}{(e+1)!} \cdot \sum_{\text{key } k} \mathrm{pr}(k) \cdot \left(\sum_{i=0}^{B-1} \epsilon^{e+1}_{\mathrm{norm}_i(k),\mathrm{norm}_i(\mathrm{ct})} \right) \end{aligned}$$

by the power means inequality. By our approximation above, this is about

$$p_e(\mathrm{ct}) \lesssim \frac{B^e}{(e+1)!} \cdot \sum_{i=0}^{B-1} \sum_{m=0}^{5dD} \gamma_m \cdot \epsilon^{e+1}_{m,\mathrm{norm}_i(\mathrm{ct})}$$

for an overall total error estimate of

$$p_e \lesssim \frac{B^e}{(e+1)!} \cdot \sum_{i=0}^{B-1} \sum_{n=0}^{5dD} \sum_{m=0}^{5dD} \gamma_m \cdot \delta_{i,n} \cdot \epsilon^{e+1}_{m,n}$$

B.4 Correlation

However, we still need to deal with correlation. We were not able to analyze expected correlation between all pairs of coefficients. We believe that the effect of correlation will be small (after the above correction) for everything except "opposite" bits, i.e. those separated by exactly D positions. Recall again our equation for multiplication modulo $\phi^2 - \phi - 1$:

$$(a + b\phi) \cdot (c + d\phi) = e + f\phi := ac + bd + (ad + bc + bd)\phi$$

Here we see that the equations for e and f share a term bd, which greatly increases their correlation, so that if $\sum e$ crosses the error threshold $x/8$, it is more likely that $\sum f$ will as well.

To bound the effect of this correlation, we calculated the probability that

$$\left|\sum(2 \cdot e + 1 \cdot f)\right| > (2 + 1) \cdot x/8$$

for if it does not, then $\sum e$ and $\sum f$ cannot both be greater than $x/8$. (The coefficients 2 and 1 provide the tightest bound.) Call this probability $\eta_{m,n}$. Experimentally, $\eta_{m,n}$ is much larger than $N\epsilon_{m,n}^2$, so the probability of 3 errors is dominated by that of one single error and one double error in opposite coefficients.

Instead of approximately $B^3/3!$ configurations, there are $B/2$ ways to have a double error and $(B-2)/2$ ways to have a distinct single error, for a total of less than $B^2/2$ configurations. So a more accurate estimate for the error probability after correcting up to 2 errors is

$$p_3 \lesssim \frac{B}{2} \cdot \sum_{i=0}^{B} \sum_{m=0}^{5d \cdot D} \sum_{n=0}^{5d \cdot D} \gamma_m \cdot \delta_{i,n} \cdot \epsilon_{m,n} \cdot \eta_{m,n}$$

Since we do not model a cost for queries, the attacker's effort is $1/p_3$. We estimated $1/p_3$ using the above approximation and entered it into Table 2.

B.5 Quantum attacks

Finally, there is the possibility that an attacker may use Grover's algorithm to find ciphertexts with large norm, or which might otherwise be more likely to cause failures. Suppose the attacker targets classes of ciphertexts – in this case, of ciphertext norms – that cause failure with at least some probability q; and that a given ciphertext has a probability p to be in those classes. If a given class of ciphertexts appears with probability p_{ct} has a probability q_{ct} to cause an error, then the attacker's work per query would be about $\sqrt{p}$ and his probability of success per query would be

$$\sum_{q_{\text{ct}} \geq q} \frac{p_{\text{ct}}}{p} \cdot q_{\text{ct}}$$

for a total probability of success per unit effort of

$$p_{\text{gr},e} := \sum_{q_{\text{ct}} \geq q} \frac{p_{\text{ct}}}{p} \cdot q_{\text{ct}} \sqrt{p}$$

We again apply power means to obtain

$$\begin{aligned} p_{\text{grover},e} &\leq \sqrt{\sum_{q_{\text{ct}} \geq q} \frac{p_{\text{ct}}}{p} \cdot (q_{\text{ct}} \sqrt{p})^2} \\ &= \sqrt{\sum_{q_{\text{ct}} \geq q} p_{\text{ct}} \cdot q_{\text{ct}}^2} \\ &\leq \sqrt{\sum_{\text{all ct}} p_{\text{ct}} \cdot q_{\text{ct}}^2} \end{aligned}$$

Squaring and expanding p_{ct} and q_{ct} as above, we then obtain

$$\begin{aligned} p_{\text{grover},3}^2 &\leq \sum_{\text{all ct}} p_{\text{ct}} \cdot q_{\text{ct}}^2 \\ &\approx \sum_{\text{all ct}} p_{\text{ct}} \left(\frac{B}{2} \cdot \sum_{i=0}^{B-1} \sum_{m=0}^{5dD} \gamma_m \cdot \epsilon_{m,\text{norm}_i(\text{ct})} \cdot \eta_{m,\text{norm}_i(\text{ct})} \right)^2 \\ &\leq \sum_{\text{all ct}} p_{\text{ct}} \cdot \frac{B^3}{4} \cdot \sum_{i=0}^{B-1} \sum_{m=0}^{5dD} \gamma_m \cdot (\epsilon_{m,\text{norm}_i(\text{ct})} \cdot \eta_{m,\text{norm}_i(\text{ct})})^2 \\ &= \frac{B^3}{4} \cdot \sum_{i=0}^{B-1} \sum_{m=0}^{5dD} \delta_{i,n} \gamma_m \cdot (\epsilon_{m,n} \cdot \eta_{m,n})^2 \end{aligned}$$

Errors corrected	0	soft	1	2	3	4	5	6
Max variance in 32nds	9	12	13	16	18	20	22	24
Quantum security	201	209	212	218	222	225	229	232

Table 7: Effectiveness of forward error correction in MamaBear.

We computed this and took the square root to recover the estimated effort for the CCA attack in Table 2.

B.6 Effectiveness of error correction

To investigate the effectiveness of error correction, we designed variants of MamaBear, but with different variance and different error correction levels. We compared what variance would give us at least 128 bits of security against quantum CCA attacks, and what the SVP hardness would be for passive attacks at that variance. We included BCH codes which correct n errors at the cost of $9n$ bits. The results are shown in Table 7. We can see that our Melas FEC adds about 17 bits of security.

We also included a soft parity code. This code adds a parity bit for each 64-bits quadrant of the key. We consider a decoded bit "doubtful" if its coefficient is within a certain numerically-optimized distance of decoding in the opposite way. Then if the parity bit indicates an error, all doubtful bits in the quadrant are flipped. This is easier to implement in constant time than finding the most-doubtful bit. Using quadrants mitigates the correlation problems listed above, because correlated bits are in different quadrants, and works with any $D \geq 260$.

B.7 Future work

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. Furthermore, there is the possibility that some sort of "fuzzy Grover" sampling algorithm could do

better than our Grover attack with a hard threshold. These attacks may further reduce security against failure. We leave analysis of this problem to future work.

C Why no Targhi-Unruh δ?

In Section 4.2, we recommended not to use the Targhi-Unruh δ tag. It adds a small amount of complexity and bandwidth, and we believe that it adds no security. Here we sketch of why. [[**TODO: turn this into a real proof**]]

Recall that **EncapsCCA** is given a public key $\mathrm{pk} = s, A$ and chooses a random seed k_0. It then computes

$$c, \delta, k_1 \leftarrow H(\mathrm{pk}||k_0)$$

It uses c along with pk to produce the KEM and DEM values B and D respectively, sends δ in the clear, and outputs k_1 as the final key. On decryption, δ isn't used directly. But **DecapsCCA** recovers k_0 and re-runs **EncapsCCA** to check that the capsule was formed honestly, including the tag δ.

Intuitively, δ serves little purpose, because the adversary is passing superpositions of values $(\mathrm{pk}||k_0)$ to the random oracle to compute c anyway. If ciphertexts have different c values, then they have different B values with overwhelming probability. Since c must match on re-encryption, checking that δ must match is redundant.

But δ gets around a problem in extending the random oracle model (ROM) security proof to the quantum world. Classically, a CCA simulator can just check if any oracle query $H(pk||k_0)$ would have produced a given ciphertext, so with high probability if the ciphertext is valid, the simulator already knows k_0. But the quantum oracle queries are superpositions, and there's no way to extract k_0 from them. Indeed the adversary might not know k_0. However, the random oracle is quantum-indifferentiable from a random

polynomial whose degree is at least twice the number of oracle queries the adversary makes [35]. In that case, the simulator can find k_0 since it's a root of the polynomial.

However, we think that the proof is patchable to work without δ, at least in the case of ThreeBears. Recall that c is expanded using the random oracle in order to sample noise vectors b and ϵ_b, and to compute

$$B = M^\top \cdot b + \epsilon_b$$

where M is derived from pk. Therefore, c controls the low bits of the $d \times D$ coefficients of B. The simulator can choose a random linear transform

$$T : (\mathbb{Z}/2\mathbb{Z})^{d \times D} \rightarrow: (\mathbb{Z}/2\mathbb{Z})^{|k_0|}$$

Instead of the protocol sending δ, the simulator can first choose b, and sample ϵ_b from χ_{σ^2} under the constraint that T(low bits of B) $= \delta$. The sampling algorithm should be efficient if σ^2 isn't too small. This construction should be quantum-indifferentiable from a random oracle.

In summary, if the simulator controls the random oracle, it has enough power to leak k_0 through ciphertexts without requiring the tag δ. We believe that δ also isn't necessary for systems like Kyber that use rounding, but the argument for why may need modification.