Module-LWE key exchange and encryption: The three bears

Mike Hamburg*
Draft; expecting to add Hart Montgomery as co-author

July 8, 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 generalized 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 [31]. Consequently, there is a growing interest in developing a suite of "post-quantum" algorithms which would resist attack by these computers [24]. The most common approaches to addressing this threat rely on the hardness of lattice problems, including variants such as learning with errors (LWE) [27], ring learning with errors (rLWE) [23], and module learning with errors (mLWE) [22].

Overall, rLWE-based schemes [16, 15, 2] tend to be faster and have smaller public keys and ciphertexts than those based on classical LWE [5].

*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 [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 [6].

1.1 Our contribution

Here we propose a new cryptosystem based on 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.

Our cryptosystem is called ThreeBears, because its modulus has the same shape as the one in Ed448-Goldilocks [13].

There are few systems based on lattices modulo generalized Mersenne numbers. Though previous proposals [1] have met with limited success [4], we believe that combining these lattices with state-of-the-art key exchange protocols will at least be worth analyzing.

We also further explore Saarinen's error correction trick [29, 28]. 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^{-55} for our weakest recommendation.

But for public-key encryption, failures may lead to a chosen-ciphertext attack [18]. 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 generalized 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, generalized 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 of degree D. They also specify one or more noise distributions χ_i over R. Typically the noise is defined as

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

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 [3], 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 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 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:

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

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" 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 pseudo-Mersenne rings

Both polynomial rings and generalized 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, generalized Mersenne rings have a different ring structure, and when N is prime they have no subrings. This may improve or weaken security. Gu's concurrent work [10] shows that the two ring structures reduce to each other with a factor of D noise increase. That factor is too large for the reduction to apply in practical systems, but it suggests that their security probably isn't too different.

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 [3].

Generalized Mersenne numbers support multiplication algorithms such as Karatsuba-Ofman [20] or Granger-Moss [11], as well as Solinas reduction [32]. 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 generalize 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, 6].

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 generalized 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 [16, 3] but not in variants of the Ding [19] and Peikert [25] protocols. Our protocol is similar to Ding and Peikert's protocols, and doesn't need division.

Summary Overall, we believe that generalized 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 [12], 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 [30, 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, where $\sigma^2 \leq 1/2$ is the variance. Let

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

Over the module, we will use the error distribution

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

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.

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 $\text{clar} \in R^*$ 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 $\text{clar} = 2^{1560} - 1$. Likewise, for $R = \mathbb{Z}/(2^{2160} - 2^{1080} - 1)\mathbb{Z}$, we set $\text{clar} = 2^{1080} - 1$.

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

$$U_1(s,i,j) := \text{decode}(\text{SHAKE-256}(s || [d,i,j];\ \text{length} = \log_2 N))$$

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, SHAKE-256 can be replaced with Strobe [14]'s PRF operation.

3 Ephemeral key encapsulation

Generalized 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 [19] key encapsulation mechanism (KEM), as shown in Figure 1. For this KEM, Alice and Bob's secrets are ephemeral. For each instance of the protocol, they must choose new secrets at random, never to be reused. The mechanism consists of three algorithms: **Keygen**, **Encaps** and **Decaps**.

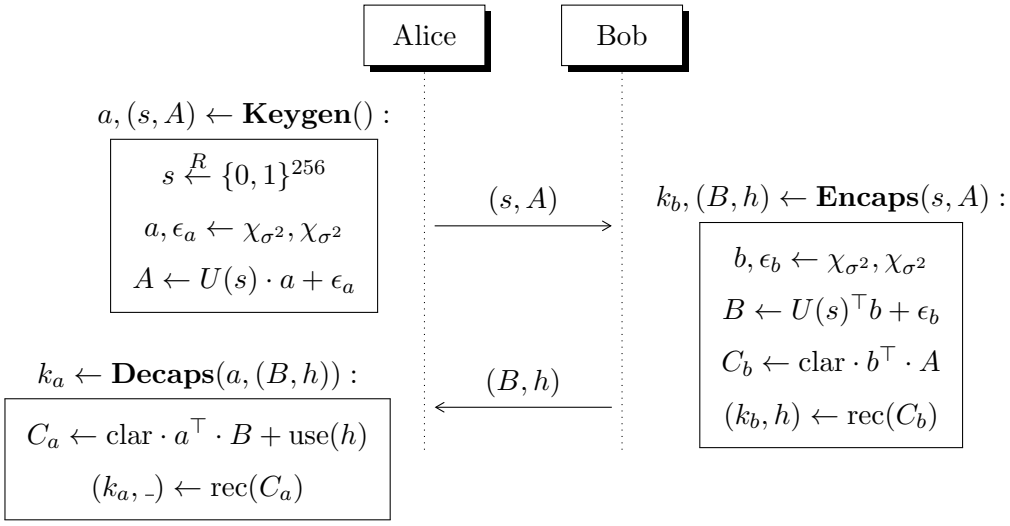

Figure 1: Ephemeral key exchange

3.1 Keygen

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.

Keygen first chooses a uniformly random seed $s \xleftarrow{R} \{0,1\}^{256}$, which is expanded to a $d \times d$ matrix $U(s)$. It then chooses vectors a and ϵ_a independently from χ_{σ^2}. The public key is

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

and the private key is a.

3.2 Encaps

The **Encaps**(s, A) algorithm creates and encapsulates an n-bit shared secret (with an even $n \leq D$) using the public key (s, A). It first chooses secrets $b \leftarrow \chi_{\sigma^2}$ and $\epsilon_b \leftarrow \chi_{\sigma^2}$, and computes

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

Encaps then computes $\text{rec}_n(C_b)$ (where the output key length $n = 256$) as follows. Let C_b be written as

$$C_b = \sum_{i=0}^{D-1} c_i \cdot x^i$$

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 $x/4$). The ring dimension is larger than the desired key length, so we will use only some of these bits, according to

$$\text{rel}_n := [0, n/2) \cup [D - n/2, D)$$

Here we take are taking the beginning and end coefficients because they have a slightly lower error rate than the middle. The shared secret and help values are then $K_b := [[(K_b)_i]]$ and $h := [[h_i]]$, respectively, for $i \in \text{rel}_n$. The output of **Encaps** is the shared secret K_b and the capsule (B, h).

3.3 Decaps

The decapsulation algorithm **Decaps**$(a, (B, h))$ extracts the shared secret from the capsule and private key. It first expands the help value to

$$\text{use}(h) := \sum_{i \in \text{rel}_n} \frac{1 - 2 \cdot h_i}{8} \cdot x^{i+1}$$

does this by likewise computing

$$C_a := a^\top \cdot B + \mathrm{use}(h)$$

It then computes K_a by the same $\mathrm{rec}_{256}(C_a)$ as in **Encaps**. The resulting key K_a agrees with the output K_b of **Encaps** with high probability. See Appendix C for correctness criteria, and Appendix D for an analysis of the failure probability.

4 CCA-secure KEM

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, by using a variant of the Fujisake-Okamoto conversion [**?**]. We will need to create algorithms **EncapsCCA** and **DecapsCCA**. The key generation is the same as for CPA-secure KEM.

4.1 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^{-55} for BabyBear[1]. 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 approach of using error correcting codes for forward error correction (FEC) [29, 28].

We use a Melas-style BCH$(511, 493, 5)$ code [21], 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

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

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.

4.2 Keygen

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.

4.3 EncapsCCA

To encapsulate a message with CCA security, we run **EncapsCCA**(s, A). This is a variant of **Encaps** which samples from $\chi_{\sigma^2} \times \chi_{\sigma^2}$ pseudorandomly instead of randomly using a hash function. The pseudorandom $\chi_{\mathrm{samp}}(s)$ is specified concretely in Appendix **??**.

[[TODO: Implement and test]] [[TODO: Make sure the security proof goes through]]

To encrypt a message m with the public key (s, A), **EncapsCCA** first chooses a random 256-bit key β uniformly at random. It expands β to two subkeys

$$(s_b, k_b) \leftarrow H(\mathrm{pk}_a || \beta)$$

Here s_b will be the seed to the sampler, and k_b will be the encapsulated key. We then proceed as in the KEM:

$$\begin{aligned} b, \epsilon_b &\leftarrow \hat{\chi}_{\mathrm{samp}}(s_b) \\ B &\leftarrow U^\top(s) \cdot b + \epsilon_b \\ C_b &\leftarrow \mathrm{clar} \cdot b^\top \cdot A \\ (m_b, h) &\leftarrow \mathrm{rec}_{256+\mathrm{feclen}}(C_b) \end{aligned}$$

We then encrypt β with forward error correction

$$\gamma \leftarrow m_b \oplus (\beta || \text{fec\_compute}(\beta))$$

Thus the masking key m_b thus must be feclen bits longer than the seed β. The capsule is then

$$(B, h, \gamma)$$

and the shared secret key is k_b.

[[TODO: I'm not sure I believe that Targhi-Unruh [33, 17] adds security. Need to look carefully at their paper. Current status: disbelieve!]]

4.4 DecapsCCA

The **DecapsCCA** algorithm recovers a shared secret from the capsule $C := (B, h, \gamma)$ and the private key (a, pk_a). It begins as in **Decaps**:

$$\begin{aligned} C_a &\leftarrow \text{clar} \cdot a^\top \cdot B + \text{use}(h) \\ (m_a, \_) &\leftarrow \text{rec}_{256+\text{feclen}}(C_a) \end{aligned}$$

It then recovers β by

$$\beta_a \leftarrow \text{fec\_correct}(m_a \oplus \gamma)$$

It then checks that **EncapsCCA**(β) produces the same capsule C. If so, it returns the secret key k_b produced by **EncapsCCA**. If not, decapsulation fails.

4.5 Recommended and toy parameters

We chose three recommended sets of parameters: BabyBear, MamaBear and PapaBear. These algorithms all exchange 256-bit keys. They are designed with estimated security of 128, 192 and 256 bits, respectively, against classical attacks. They are designed with a goal of roughly 128-bit security against quantum attacks, since they can only transport 256-bit keys, but with increasing security margins.

Our primary recommendation is MamaBear. The smaller BabyBear does not quite meet the security goal of 128 bits against quantum failure attacks. The stronger PapaBear is probably overkill, but would be a useful hedge against improvements in cryptanalysis.

We also chose three "toy" sets, intended to stimulate cryptanalysis: GummyBear, TeddyBear and DropBear. We compare our parameters to related work in Table 1.

We tuned the variance of the noise in these systems 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.

System	Toy?	Ref	d	d_{total}	σ^2	q
GummyBear	Y	This paper	1	270	10/64	256
TeddyBear	Y	This paper	1	390	8/64	256
DropBear	Y	This paper	2	540	5/64	256
BabyBear		This paper	2	624	11/32	1024
MamaBear		This paper	3	936	7/32	1024
PapaBear		This paper	4	1248	5/32	1024
JarJar	Y	[2]		512	12	12289
Kyber		[6]		768	2	7681
NewHope		[2]		1024	8	12289
trunc8		[29]		512	23.6	12289
Hila5		[28]		1024	8	12289
NTRUEncrypt		[16]		743	$\approx 2/3$	2048
NTRU Prime		[3]		739	0.28	9829

Table 1: Parameters for ThreeBears and related work.

4.6 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 corre-

spond 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 [30, 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**]]

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.

[[**TODO: Reconcile our numbers for NTRU Prime, and DJB's and Kyber's**]] [[**TODO: NTRU KEM https://eprint.iacr.org/2017/667**]] [[**TODO: I-RLWE https://eprint.iacr.org/2017/641**]]

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 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 com-

	lattice security			failure security			Message bytes		
System	C	Q	P	Bit	F	G	Pub	KEM	Enc
GummyBear	54	49	38	32	72	59	302	302	339
TeddyBear	82	75	58	34	77	64	422	422	460
DropBear	112	102	80	51	111	92	572	572	609
BabyBear	142	128	100	55	136	122	812	812	850
MamaBear	214	194	151	83	200	174	1202	1202	1240
PapaBear	284	258	201	112	263	225	1592	1592	1630
JarJar	131	118	92	55	-	-	928	1024	-
Kyber	178	161	126	142	142	-	1088	-	1184
NewHope	281	255	199	61	-	-	1824	2048	-
trunc8	141	131	102	13	45	-	1024	1024	-
Hila5	281	255	199	27	135	-	1824	-	2012
NTRUEncrypt	176	159	125	112	112	-	1022	-	1022
NTRU Prime	139	126	99	∞	∞	∞	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.

pletely 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 security margin against these improvements in MamaBear and PapaBear.

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 toy bears face another possible attack: due to the tiny amount of noise per coefficient, their private keys simply don't have enough entropy. Even DropBear's keys have only 255 bits of entropy, which expose it to 128-bit attack by Grover's algorithm or possibly even a classical meet-in-the-middle attack. Our recommended security parameters do not share this problem.

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.

5 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 [13]. More levels would be faster, but we didn't do that in this reference version.

We benchmarked our code on several different platforms. The results are shown for an Intel Core-i3-6100U (Skylake) in Table 3; on ARM Cortex-A53 in Table 4; and on 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 our performance to the `NewHope`'s reference C code. As might be expected, on 64-bit platforms, all three bears are faster than `NewHope`'s reference code, but they lose ground on 32-bit platforms. This partially because ThreeBears uses multi-precision arithmetic, and partially because it uses the slow, 64-bit SHAKE-256 everywhere instead of the faster, 32-bit ChaCha20. This can also be seen in comparison to Kyber, which uses SHAKE-128.

System	Keygen	KEM	DeKEM	Encrypt	Decrypt
BabyBear	75k	92k	19k	102k	124k
MamaBear	147k	171k	27k	185k	215k
PapaBear	241k	275k	35k	292k	330k
NewHope ref	259k	385k	73k	-	-
Kyber ref	265k	-	-	322k	364k

Table 3: Performance of reference code in cycles on a NUC with Intel Core i3-6100U "Skylake" 64-bit processor at 2.3GHz. This processor doesn't have TurboBoost. Compiled with `clang-3.9 -O2 -DNDEBUG -march=native` [[**TODO: try also gcc-7?**]]

System	Keygen	KEM	DeKEM	Encrypt	Decrypt
BabyBear	184k	240k	61k	256k	321k
MamaBear	368k	451k	86k	473k	564k
PapaBear	614k	724k	111k	751k	867k
NewHope ref	589k	913k	236k	-	-
Kyber ref	550k	-	-	751k	921k

Table 4: Performance of reference code 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`

System	Keygen	KEM	DeKEM	Encrypt	Decrypt
BabyBear	618k	769k	161k	851k	1022k
MamaBear	1243k	1469k	235k	1585k	1832k
PapaBear	2083k	2382k	309k	2522k	2846k
NewHope ref	1026k	1552k	377k	-	-
Kyber ref	1514k	-	-	1939k	2152k

Table 5: Performance of reference code in cycles on a BeagleBone Black with Cortex-A8 32-bit processor at 1GHz. Compiled with `clang-3.9 -Os -DNDEBUG -mcpu=cortex-a8 -mthumb`

[[**TODO: rerun**]]

[[**TODO: Optimized numbers; reflect updates in sampling and clarification; consider AES or ChaCha20 sampler; memory**]]

5.1 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.**]]

6 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 [24]. 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 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. 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.

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] 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.

[5] 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.

[6] 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.

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

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

[12] 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.

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

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

[15] 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`.

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

[17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, Report 2017/604, 2017. `http://eprint.iacr.org/2017/604`.

[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] 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`.

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

[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. *Journal of the ACM (JACM)*, 60(6):43, 2013.

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

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

[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] 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`.

[29] 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.

[30] 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.

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

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

[33] 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

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

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 $2^{12 \cdot 268+1} - 1$, 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}.

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 number would be secure if it had no sparse factors. We could even use a Fermat number, but unfortunately these tend to have at least somewhat sparse factors.

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, $\Phi_{2 \cdot 607 \cdot 13}(2)$ works with clarifier $2^{280 \cdot 13+1} - 2^{140 \cdot 13} - 1$. We did not see an appreciable gain in this approach, 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 generalized 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.

A.1 Other noise distributions

Our ring and noise distribution result in uneven error 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 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 product of the primitive polynomials $(p_1, p_2) := (t^9 + t^4 + 1, t^9 + t^5 + 1)$ by rotating and xoring. This gives an 18-bit FEC value, as shown in Listing 1.

```
u32 compute(const u8 *data, unsigned len) {
    u32 r=0;
    for (unsigned i=0; i<len; i++) {
        r ^= data[i];
        r = (r>>4) ^ (r&0xF)*0x4221 ^ (r&0xF)*0x402;
        r = (r>>4) ^ (r&0xF)*0x4221 ^ (r&0xF)*0x402;
```

```
    }
    return r;
}
```

Listing 1: Code to compute Melas FEC

To add error correction, we simply append 3 bytes of the result of `melas_compute`. This makes the protected data a code-word, meaning that

$$\texttt{melas\_compute}(\text{data}||\text{fec}) = 0$$

To decode, we calculate a syndrome `melas_compute`(data) and reduce it mod p_1 and p_2 to calculate $(s_1, \hat{s}_{-1})$. Let $s_{-1} :=$ bit-reverse$(\hat{s}_{-1})$. We will then work modulo p_1. Let's assume for now that there are two errors at position e_1 and e_2, and let

$$E_1 := t^{e_1} \bmod p_1, \quad E_2 := t^{e_2} \bmod p_1$$

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 p_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.

```
u32 mul(u32 a, u32 b) {
    // Montgomery multiply a*b
    u32 r = 0;
    for (unsigned i=0; i<9; i++) {
        r ^= ((b>>8)&1) * a;
        b <<= 1;
        a = a>>1 ^ (a&1)*(0x221>>1);
    }
    return r;
}

u32 div_t8n(u32 x, int n) {
    // Return x / t^8n
    for (n*=2; n>0; n--)
        x = (x>>4) ^ (x&0xF) * 0x21;
    return x;
}

void correct(u8 *data, unsigned len) {
    u32 a,b,c,i,j;

    // Montgomery reduce syndrome mod polynomials
    for (i=0, a=b=compute(data,len); i<5; i++) {
        b = b>>3 ^ (b&7)*(0x221>>3);
        a = a>>3 ^ (a&7)*(0x211>>3);
    }
    // Bit reverse b
    c = (b^(b>>2)) & 0x49; b^=(c|c<<2);
    c = (b^(b>>6)) & 0x07; b^=(c|c<<6);
```

```
    gf_t r=c=mul(a,b), s=0;
    // s = Half-trace(1/c)
    for (i=0; i<7; i++) r = mul(mul(r,r),c);
    const u8 table[9] =
        {14,60,91,223,37,8,47,223,36};
    for (i=0; i<9; i++) s ^= ((r>>i)&1)*table[i];
    a = div_t8n(a,61-len); // Adjust for length
    s = mul(a,s);

    // Correct the errors
    for (j=0; j<2; j++, s^=a) {
        for (i=0, r=s; i<len; i++) {
            r = div_t8n(r,1);
            gf_t mask = ((r & (r-1))-1)>>9;
            data[i] ^= r & mask;
        }
    }
}
```

Listing 2: Code to correct Melas FEC

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

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

and

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

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

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

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 $e_i - c_i$.

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 (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 $c = d = \pm 1$.

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 nearly identical, 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[2] 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

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

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

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

The norm always at most $5dD$. It can attain this only for $i = D/2$; for other coefficients it tapers down to a maximum of $2dD$ at $i = 0$.

For each norm $n \leq 5 \cdot d \cdot D$, we computed the distributions of the noise in the output. With our rounding mechanism, the error probability ramps

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

from 0 at $x/8+1$ to 1 at $3x/8-1$, and the conditional probability of a single bit error is

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

Likewise, for each norm $n < 5dD$ 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 $B := \min(D, 256+18)$ 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.

D.1 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, 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 approxmately

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

for an overall total error estimate of

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

D.2 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 seem 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.

D.3 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

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

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

D.4 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.