Integer Module LWE key exchange and encryption: The three bears

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

July 19, 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 [29]. Consequently, there is a growing interest in developing a suite of "post-quantum" algorithms which would resist attack by these computers [23]. The most common approaches to addressing this threat rely on the hardness of lattice problems, including variants such as learning with errors (LWE) [25], ring learning with errors (RLWE) [22], and module learning with errors (MLWE) [21].

Overall, RLWE-based schemes [15, 14, 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 [24]. 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 *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.

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. Gu's concurrent work [10] covers some theoretical ground but contains no concrete proposals, and Joux et al.'s system [1] offers less security than intended [4]. We propose ThreeBears as a starting point for practical analysis.

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 [16]. Saarinen proposed to improve this tradeoff by using an error-correcting code [27, 26]. 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: Kyber's overall structure [6]; Ding's reconciliation [18]; Saarinen's error correction [27]; and generalized Mersenne ring structure with Goldilocks' prime shape [13]. The resulting system is competitive with other state-of-the-art RLWE KEMs in terms of simplicity, performance and conjectured security level.

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 [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 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:

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

We could instead use a fixed Hamming weight or even rounding. Again, limiting errors generally requires 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 [10] shows that the polynomial and GM rings have equivalent security up to a factor of D noise increase in either direction. This increase makes the reduction meaningless for ThreeBears, but it suggests RLWE and I-RLWE may have similar security.

Multiplication algorithms Some polynomial rings support very fast multiplication based on the number-theoretic transform (NTT). The NTT can be performed in place, giving an intrinsic memory savings.

Other polynomial rings lack 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]. GM rings, like these non-NTT rings, can support multiplication using the naïve "schoolbook" algorithm, or faster algorithms like Granger-Moss [11] or Karatsuba-Ofman [19] with Solinas reduction [30]. However, GM rings have to propagate carries whereas polynomial rings have to reduce each coefficient; which is more expensive depends on platform.

Processor support Polynomial rings typically have 10- to 14-bit q, so they do arithmetic with 16-bit `shorts` 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.

Generalized Mersenne fields can use limbs as large as the CPU's multiplier, so they are fast on large scalar CPUs and with existing cryptographic hardware accelerators. They are also supported by existing multi-precision arithmetic libraries.

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. This is a logical choice in a GM ring. For a polynomial ring, most systems use a prime q, but some (such as NTRU [15]) use a power of 2.

Error amplification Non-cyclotomic rings amplify noise when reducing modulo P or N. This results in slightly worse performance and security. Cyclotomics are a natural choice for polynomial rings, but not as natural for GM rings. Our rings amplify errors by a factor of up to 3/2.

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 [12], so a 256-bit key is appropriate for high-security systems.

Accordingly, 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 [28, 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.

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

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

System	d	$d \cdot D$	σ^2	x	Failure prob. Eph	Failure prob. CCA	PQ Security Lattice	PQ Security CCA
BabyBear	2	624	11/32	1024	2^{-58}	2^{-141}	128	126
MamaBear	3	936	8/32	1024	2^{-71}	2^{-171}	197	151
PapaBear	4	1248	7/32	1024	2^{-70}	2^{-170}	270	152

Table 1: Main recommendations, failure probabilities, and log security estimates against quantum attacks.

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 coefficient of each ring element.

2.6 Recommended parameters

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

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

The 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. Our main recommendation is MamaBear.

We analyze other parameter choices in Appendix A, including different rings and toy parameters.

2.7 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 chosen to minimize the variance of the coefficients of

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

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

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

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

2.8 Uniform distribution from seed

For our key exchange, we need the two parties to agree on a public, random $d \times d$ square matrix over R. Like NewHope and Kyber, we do this by expanding from a seed according to some function

$$U : \{0,1\}^{\text{seedlen}} \to R^{d \times d}$$

which we model as a random oracle. Internally, U uses cSHAKE-256.

3 Ephemeral key encapsulation

GM rings are suitable for many of the same protocols that polynomial rings are used for. We begin by describing a Ding-like [18] key encapsulation mechanism (KEM), as shown in Figure 1. For this KEM, Alice and Bob's secrets are ephemeral and must never be reused. The mechanism consists of three algorithms: **KeygenEph**, **EncapsEph** and **DecapsEph**.

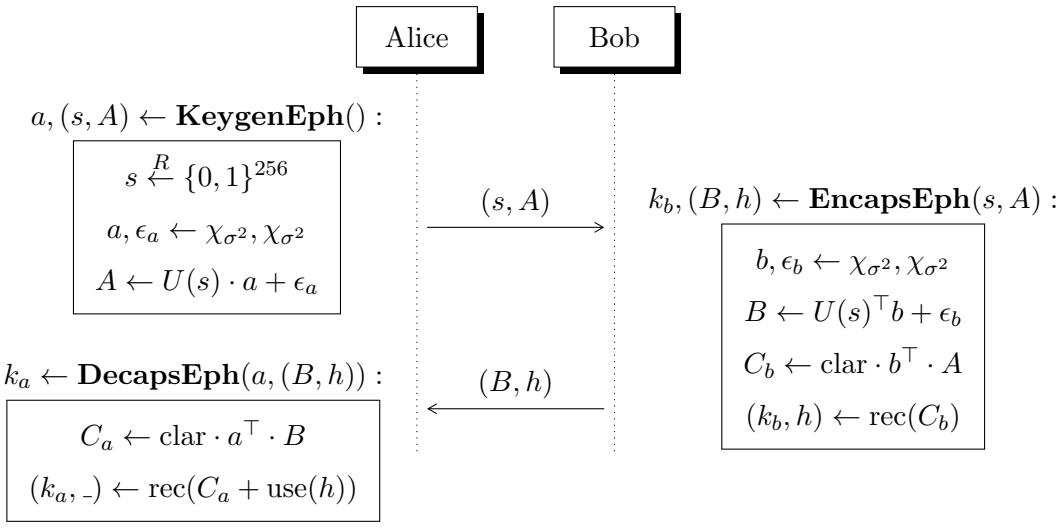

Figure 1: Ephemeral key exchange

3.1 KeygenEph

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.

KeygenEph 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 EncapsEph

The $\mathbf{EncapsEph}(s, A)$ algorithm creates and encapsulates an n-bit shared secret using the public key (s, A). We require that $n \leq D$, and in practice we will use $n = 256$. **EncapsEph** first chooses secrets $b \leftarrow \chi_{\sigma^2}$ and $\epsilon_b \leftarrow \chi_{\sigma^2}$, and computes

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

EncapsEph then computes $\mathrm{rec}_n(C_b)$ as follows. Let C_b be written as

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

Let

$$c_i = \frac{x}{2}(K_b)_i + \frac{x}{4}h_i + \mathrm{small}_i \quad \text{where} \quad 0 \le \mathrm{small}_i < \frac{x}{8}$$

That is, $(K_b)_i$ and h_i are the top two bits of c_i. The ring dimension is larger than the desired key length, so we will use only some of these bits, according to

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

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

3.3 DecapsEph

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

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

and computes

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

It then computes K_a by the same $\mathrm{rec}_n(C_a)$ as in **EncapsEph**. The resulting key K_a probably agrees with the output K_b of **EncapsEph**, but with some small probability the key exchange will fail. See Appendix B for an analysis of the failure probability.

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 [31]. This will define algorithms **KeygenCCA**, **EncapsCCA** and **DecapsCCA**

4.1 Forward error correction

As with any LWE-based cryptosystem, the security and failure probability of the system are both functions of the noise distribution. We chose noise that puts the failure probability in an appropriate range for key exchange, around 2^{-58} 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) [27, 26].

We use a Melas-style BCH$(511, 493, 5)$ code [20], 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 bits longer. For simplicity, we round both of these overheads up to 3 bytes. Our Melas code is fast, constant-time, and moderately complex — some 65 lines of C.

Since our larger N gives a 312-bit key, it has room for up to 56 bits of error correction. We would rather use a larger code which corrects more errors, but a larger BCH code might be too complicated to correct 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.

4.2 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. Also, when generating keys using a pseudo-random generator, the generator should be domain separated from ephemeral keygen to prevent mistakes.

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

4.3 EncapsCCA

To encapsulate a message with CCA security, we run **EncapsCCA**(s, A). This is a variant of **EncapsEph** which samples from $\chi_{\sigma^2} \times \chi_{\sigma^2}$ pseudorandomly using a hash function[2] instead of randomly. Call this pseudorandom sampler $\chi_{\text{samp}} : \{0,1\}^{256} \to R^d \times R^d$.

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 three subkeys, each 256 bits long:

$$(s_b, \delta, k_b) \quad \leftarrow \quad H(\text{pk}_a || \beta)$$

Here s_b will be the seed to the sampler, and k_b will be the encapsulated key. The value δ is used only by the Targhi-Unruh proof [3]. We then proceed as in the KEM:

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

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

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

and the shared secret key is k_b.

[[TODO: make sure the proof goes through. Check if hashing key is needed. Try to improve Targhi-Unruh to get rid of δ.]]

[2] That is, according to a random oracle.

[3] The δ value appears to be purely a proof artifact in Targhi-Unruh, and it is clearly redundant in a classical context. Hopefully it will be obviated by an improved proof for CCA-secure KEMs, but for now we include it out of caution.

4.4 DecapsCCA

The **DecapsCCA** algorithm recovers a shared secret from the private key (a, pk_a) and a capsule (B, h, γ, δ). It begins as in **DecapsEph**:

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

It then recovers β by

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

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

4.5 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 [12]. 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 [28, 8]. We estimated the "core quantum SVP hardness" of this attack using NewHope's BKZ parameter estimation scripts. 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			Message bytes	
System	Lattice	CCA	Failure	PK	Capsule
BabyBear ephem	128	-	58	812	812
MamaBear ephem	197	-	71	1202	1202
PapaBear ephem	270	-	70	1592	1592
BabyBear CCA	128	126	141	812	882
MamaBear CCA	197	151	171	1202	1272
PapaBear CCA	270	152	170	1592	1662
Kyber light [6]	102	169^c	169	736	832
Kyber rec. [6]	161	142^c	142	1088	1184
Kyber paranoid [6]	218	145^c	145	1440	1536
JarJar [2]	118	-	55	928	1024
NewHope [2]	255	-	61	1824	2048
trunc8 [27]	131	-	45	1024	1024
Hila5 [26]	255	135^c	135	1824	2012
NTRU ees743p1 [15]	159	-	112	1022	1022
NTRU Prime [3]	126	∞	∞	1232	1141
NTRU KEM [17]	123	∞	∞	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. Columns marked ^{c} are against a classical adversary.

[[TODO: Reconcile our numbers for NTRU Prime, and DJB's and Kyber's]]
[[TODO: NTRU KEM https://eprint.iacr.org/2017/667]] [[TODO: Private key size?]]

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.

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 $64 \times 64 \to 128$-bit multiplication when the compiler and CPU support them.

Our reference code caches the expanded private key, which costs $416 \cdot d$ bytes on a 64-bit platform and $480 \cdot d$ bytes on a 32-bit platform. For CCA-secure decapsulation, it also caches the public key (which is needed to re-encrypt) but not the matrix $U(s)$.

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 our performance to the `NewHope` and `Kyber`'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 cSHAKE-256 everywhere instead of the faster, 32-bit ChaCha20. This can also be seen in comparison to Kyber, which uses cSHAKE-128.

[[**TODO: Optimized numbers**]] [[**TODO: Actually deploy cSHAKE**]]

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.**]]

		Ephemeral		CCA-secure	
System	Keygen	Encaps	Decaps	Encaps	Decaps
BabyBear	75k	92k	19k	102k	124k
MamaBear	147k	171k	27k	185k	215k
PapaBear	241k	275k	35k	292k	330k
NewHope	259k	385k	73k	-	-
Kyber	265k	-	-	322k	364k

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	184k	240k	61k	256k	321k
MamaBear	368k	451k	86k	473k	564k
PapaBear	614k	724k	111k	751k	867k
NewHope	589k	913k	236k	-	-
Kyber	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	618k	769k	161k	851k	1022k
MamaBear	1243k	1469k	235k	1585k	1832k
PapaBear	2083k	2382k	309k	2522k	2846k
NewHope	1026k	1552k	377k	-	-
Kyber	1514k	-	-	1939k	2152k

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

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 [23]. 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] 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`.

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

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

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

[18] 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`.

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

[20] 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.

[21] 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`.

[22] 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.

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

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

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

[26] 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`.

[27] 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.

[28] 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.

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

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

[31] 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 Solinas 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).

We originally chose our N for degree $D = 260$ with $x = 2^{12}$, but it seems that $D = 312$ and $x = 2^{10}$ is a better choice. If our narrow noise distribution turns out to be a problem, then $D = 260, x = 2^{12}$ should be a good fallback.

A.1 Smaller rings without error correction: ThreeWolves

A final worthwhile option is $P = x^{260} + x^{130} - 1$ with $x = 2^{10}$. This is incompatible with error correction (except for soft error correction) because it has degree only 260, but it is otherwise a good choice. Reasonable parameters are given in Table 7.

We would serialize elements of this ring as 2600 bits (325 bytes), dropping the top bit if the element is at least than 2^{2600}. This has a negligible chance of happening, and would only slightly increase the noise anyway.

It was difficult to decide whether to recommend the ThreeBears or the ThreeWolves as our main proposal. The Bears' error correction

System	$d \cdot D$	σ^2	PQ Security Lattice	PQ Security Failure	Bytes PK	Bytes Eph	Bytes CCA
BabyWolf	$2 \cdot 260$	8/32	98	92	682	682	746
MamaWolf	$3 \cdot 260$	5/32	149	131	1007	1007	1071
PapaWolf	$4 \cdot 260$	4/32	202	147	1332	1332	1396

Table 6: ThreeWolves with $D = 260$, $x = 2^{10}$ and no error correction.

System	$d \cdot D$	σ^2	Security Lattice	Security Failure	Bytes PK	Bytes Eph	Bytes CCA
GummyBear	$1 \cdot 270$	10/64	$54^c/49^q$	59	302	302	371
TeddyBear	$1 \cdot 390$	8/64	$82^c/74^q$	69	422	422	492
DropBear	$2 \cdot 270$	5/64	$112^c/102^q$	92	572	572	641

Table 7: Toy bears with $x = 2^8$. Lattice security is listed against both classical and quantum adversaries, because someone might actually break these.

adds maybe 9% extra security at the expense of complexity. Due to their larger dimension, they produce oversized keys and give less granularity to tune security by changing d. On the other hand, I-MLWE is a new primitive and post-quantum security is an emerging field, so we thought the higher security estimates for the Bears made them more appropriate, and the noise level in the Wolves is a little small for comfort. The Bears are also slightly easier to implement due to N being less than a power of 2 rather than greater, and are faster for their size on 64-bit platforms.

A.2 Toy parameters

We propose toy parameters to encourage cryptanalysis. Our toy parameters have $x = 2^8$, smaller degrees and less noise. This exposes them to lattice attacks, and also another attack: due to the tiny amount of noise per coefficient, the entropy of their private keys is almost small enough to exhaust.

A.3 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 Failure probability and chosen-ciphertext attacks

B.1 Correctness criterion

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.

B.2 Failure probability without error correction

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[4] 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 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

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

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

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.

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, 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 approximately

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

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 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.

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

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

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

B.6 Effectiveness of error correction

We wish to compare the effectiveness of various degrees of error correction. To do this, we generated variants of MamaBear, but with different variance and different error correction. We compared what variance would give us at least 128 bits of security against active quantum attacks, and what the resulting security level would be against "known quantum" (Q) attacks[5]. We included BCH codes which correct n errors at the cost of $9n$ bits. The results are shown in Table 8. We can see that our Melas FEC adds about 19 bits of security.

[5]The real MamaBear has a slightly lower variance to leave a security margin, since our analysis may not be tight.

Errors corrected	0	doubt	1	2	3	4	5	6
Max variance in 64ths	9	12	13	18	20	23	26	28
Quantum security	182	189	192	201	204	208	212	214

Table 8: Effectiveness of forward error correction in MamaBear

We also included a parity-based soft error correcting code. We consider a bit "doubtful" if its coefficient is within a certain distance of decoding in the opposite way. Then if the parity bit indicates an error, all doubtful bits are flipped.[6] To mitigate the correlation problems listed above, we actually divide the key bits into 4 sections and perform the error correction on each section; since the correlated bits are in separate sections they won't contribute to the probability of an uncorrectable double error. This would allow us to correct some errors with only 4 extra bits, i.e. with $D = 260$.

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.

[6] This is obviously less effective than flipping the most-doubtful bit, but it is easier to implement in constant time.