EagleSignV3 : A new secure variant of EagleSign signature over lattices

Abiodoun Clement Hounkpevi[(1)], abiodounkpevi@gmail.com
Sidoine Djimnaibeye[(1)], dthekplus@gmail.com
Michel Seck[(2)], michelseck2@gmail.com
Djiby Sow[(1)], sowdjibab@yahoo.fr, djiby.sow@ucad.edu.sn

[1] Cheikh Anta Diop University Dakar Senegal
[2] Ecole Polytechnique Thies Senegal

Table of Contents

Abstract. With the potential arrival of quantum computers, it is essential to build cryptosystems resistant to attackers with the computing power of a quantum computer. With Shor's algorithm, cryptosystems based on discrete logarithms and factorization become obsolete. Reason why NIST has launching two competitions in 2016 and 2023 to standardize post-quantum cryptosystems (such as KEM and signature) based on problems supposed to resist attacks using quantum computers. EagleSign was prosed to NIT competition in Jun 2023 as an additional signature. An improvement called EagleSign-V2 was proposed in December 2023 but Tibouchi and Pells prove that these two variants don't hold the zero knowledge property. In this document we present the family of lattices based post-quantum signatures called EagleSignV3. They are secure and efficient successors of both EagleSign-V1 (NIST, June 2023) and EagleSign-V2 (NIST forum, December 2023). The public key of EagleSignV3 is based

on a mix of MLE (Module Learning with Error) and MNTRU (module variant of the famous NTRU problem). The instantiations EagleSignV3 are new variants of the EagleSign signatures family posted to NIST competition in June 2023 as additional signatures. EagleSignV3 uses the rejection of Lyubashevsky-2012 to achieve the zero-knowledge property. The main difference between EagleSign and Dilithium is the public key. We have two instantiations based either on ring or on module. The sizes of the ring based variant of EagleSignV3 are close to those of Dilithium but the sizes of its module based instantiation is bigger than those of Dilithium.
NB: The implementation of EagleSign-V1 is available on NIST website and those of EagleSign-V2 can be found on Github at https://github.com/EagleSignteam/EagleSign_v2 and in NIST forum as a comment on improvements on EagleSign in December 2023. The implementation of EagleSign-V3 can be deduced from those of EagleSignV2.
Keywords: Public key cryptography, Signature, Lattice, NTRU, MNTRU, LWE, MLWE, Dilithium, Falcon, EagleSign

1 Introduction

Given the recent advancements in quantum computing and the fact that the classical Integer Factorization Problem and the Discrete Logarithm Problem are not secure against quantum computers [77], the scientific community want to design cryptosystems and protocols that resist to attacks by quantum technologies.

For this reason, the National Institute of Standards and Technology (NIST), by a call for submissions [59], propose the transition to quantum-resistant cryptography. Many algorithms for public-key encryption, key encapsulation mechanism, and digital signature were proposed throughout 3 rounds. Many authors have worked on the categorization (according to the family of underlying problem) and the performance analysis of the schemes proposed to NIST [24, 37, 57, 60]. There were 3 evaluation criteria for the case of digital signature schemes: (1) security (Zero knowledge property, security proof in ROM/QROM, Side Channel Attacks mitigation, hardness of the underlying problem), (2) cost and performance, and (3) algorithm and implementation characteristics on software and hardware.
In July 2022, at the end of the 3rd round, regarding the post-quantum digital signatures, there were 3 candidates proposed for NIST standardization: one MLWE-based signature (CRYSTALS-Dilithium), one NTRU-based signature (FALCON) and one hash-based signature (Sphincs+).

At the end of the 3rd round, NIST began a new process by a call for additional signatures and the first submissions where done in June 2023 with 40 proposals including EagleSign which is a lattices based signature using a public key that is a mix of MLWE and MNTRU. Tibouchi in the PQC Forum of NIST have proposed an attack on June on the first variant called here EagleSign-V1 and in December on the second variant called EagleSign-V2. These two attacks target the zero knowledge property of EagleSign. In this work, we propose a new variant

of eagleSign called eagleSignV3 that use correctly the rejection sampling of Lyubashevsky 2012 to obtain the zero-knowledge property .

1.1 Short description of Falcon/Modfalcon and Christal Dilithium

Summary of Falcon (Ring) and ModFalcon (Module) : Falcon and its generalization ModFalcon are based on the framework for lattice-based signature schemes proposed by Gentry, Peikert and Vaikuntanathan : hash-and-sign paradigm upon collision-resistant preimage sampleable function [39]. The underlying hard problem in Falcon is NTRU-SIS (Short Integer Solution problem over NTRU public key) together with the "Fast Fourier sampling (FFT)" as a trapdoor sampler. In the ring $R_q = \dfrac{\mathbb{Z}_q(X)}{(X^n+1)}$, the NTRU public key of Falcon is $h = f^{-1}g \mod q, q = 12289, n = 512, 1024$ where f, g are small and sparse polynomials in R_q. The NTRU-SIS hardness is based on the difficulty of recovering the polynomials f and g given the polynomial ring element h. In quantum or classical world, no efficient attack is currently known to break the computational NTRU-SIS or the Decisional Small Polynomial Ratio (DSPR) assumption of NTRU whenever f and g are suitably chosen. In Falcon, after computing f and g from an appropriate distribution, the key generation algorithm computes F and G such that $fG - gF = q \mod X^n + 1$. The polynomials f, g, F, and G are stored in the private key sk.
To sign a message m, Falcon uses a hash function H, a private key sk, a salt $r, |r| = 64$ and a FFT sampler to compute short vectors s_1, s_2 that satisfy the equation: $s_1 + s_2 h = H(r, m)$. Falcon is the most compact (most small size) signature among those proposed to NIST competition but it is based directly on cyclotomic ring and does not allow various security levels.
ModFalcon is introduced by Chuengsatiansup, Prest, Stehlé, Wallet and Xagawa (ASIACCS '20) and it generalizes Falcon to modules where the public key is $\mathbf{H} = \mathbf{F}^{-1}\mathbf{G} \mod q$ where $\mathbf{F}$,(resp: $\mathbf{G}$) is $m \times m$ (resp: $m \times k$) matrix with short entries in R_q. In [29], they instantiated a particular case where $k = 1, q = 12289$, and $n = 256$. Moreover, in the IBE scheme (IACR ePrint 2019/1468) the authors Cheon, Kim, Kim and Son chose $m = 1$. ModFalcon allows an intermediate security level that is missing in Falcon signature.

Fiat-Shamir Transformation: The Fiat-Shamir transformation was proposed by Fiat and Shamir [36] as a framework that allows to derivative a signature from an Identification Protocol (ID) by removing the interaction in ID throughout a hash function. Many studies for the security of Fiat-Shamir signatures in the Quantum random oracle Model (QROM) were done (see [17, 17, 34]). Recently two important works of Barbosa *et al.* [17] and of Devevey *et al.* [26], were done to improve the security of the Fiat-Shamir signature and fix flaws in existing proofs such as in Dilithium.

Summary of Dilithium (hight level description): Crystals Dilithium is a Fiat-Shamir signature with aborts over lattices based on MLWE and MSIS hard problems which is based on Vadim Lyubashesky previous works in 2009 and 2012 [53, 54]. In Dilithium, the security of the public keys is based on MLWE and

the security of the signature against forgery is based on MSIS and SelfTargetMSIS problems. The public key with MLWE over $R_q = \frac{\mathbb{Z}_q(X)}{(X^n+1)}, q = 2^{23} - 2^{13} + 1, n = 256$ is $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ where $\mathbf{A} \in R_q^{k \times l}$ is a public matrix generated uniformly at random and the secrets $sk = (\mathbf{s}_1, \mathbf{s}_2) \in R_q^l \times \in R_q^k$ are generated uniformly at random such that $|\mathbf{s}_1|_\infty, |\mathbf{s}_2|_\infty \leq \eta$ (a short integer). To sign a message m, Dilithium uses a hash function H, the private key sk to compute an ephemeral public key $\mathbf{d} = \mathbf{A}\mathbf{y}$ (together with an ephemeral secret key $\mathbf{y}$), a sparse challenge $c = H(\mathbf{d}, m)$ and sets $\sigma = (\mathbf{z}, c, \mathbf{h})$ as signature where $\mathbf{z} = c\mathbf{s}_1 + \mathbf{y} \in R_q^l$ and $\mathbf{h}$ is a hint vector. To protect $\mathbf{z}$, a "while loop" for rejection sampling containing few steps is included in the process before a valid signature with zero knowledge property is obtained. For this, a counter is incremented in every loop to generate a different ephemeral secret key $\mathbf{y}$ in each iteration. To reduce the size of the signature a special technique based on rounding and hight bits is used. Dilithium has two variants according to the way the ephemeral secret key $\mathbf{y}$ is generated (deterministic or probabilistic).
Recently many other signatures based on NTRU/MNTRU and RLWE/MLWE were proposed [20, 62].

1.2 Summary of EagleSignV3

EagleSignV3 is a Fiat-Shamir signature with aborts over lattices. Our variant is implemented over $R_q = \frac{\mathbb{Z}_q(X)}{(X^n+1)}$ with $n = 512, 1024, 2048$ and $q = 1 \mod 2n$ in order to make NTT easy to use.
Denote $S_\eta = \{u \in R_q / |u|_\infty \leq \eta\}$ the polynomials in $\mathcal{R}_q$ whose $l\infty$ norm is tightly upper-bounded by η.
The public key over R_q (where q is a prime) is $\mathbf{T} \in R_q^{k \times l}$ where $\mathbf{T} = (\mathbf{A}\mathbf{F}^{-1} + \mathbf{D})\mathbf{G}^{-1}$, $\mathbf{A} \in R_q^{k \times l}$ is a public matrix generated uniformly at random and the secrets $\mathbf{F} \in S_{\eta_F}^{l \times l}, \mathbf{G} \in S_{\eta_G}^{l \times l}$ (resp: $\mathbf{D} \in S_{\eta_D}^{k \times l}$) are invertible matrices of small polynomials generated uniformly at random (resp: matrix of small polynomials generated uniformly at random). Note that $\mathbf{F}$ or $\mathbf{G}$ can be a constant or a polynomial suitably chosen. The secret key is then $sk = (\mathbf{F}, \mathbf{G}, \mathbf{D}) \in S_{\eta_F}^{l \times l} \times S_{\eta_G}^{l \times l} \times S_{\eta_G}^{k \times l}$. Note that, to sign a message M, EagleSignV3:

- uses two hash functions H, G (H is modelled as a random oracle in ROM security proof) and a private key sk to compute an ephemeral public key $\mathbf{p} = \mathbf{T}\mathbf{Y}_1 + \mathbf{Y}_2 \in R_q^{k \times m}$ (together with an ephemeral secret key $(\mathbf{Y}_1, \mathbf{Y}_2) \in S_{\eta_{Y_1}}^{l \times m} \times S_{\eta_{y2}}^{k \times m}$), a challenge $\mathbf{C} \in S_{\eta_c}^{l \times m}$ derived from $H(M, r)$ where $r =: G(\mathbf{P})$
- and sets $\sigma = (r, \mathbf{Z}, \mathbf{W})$ where $\mathbf{Z} = \mathbf{Y}_1 + \mathbf{GFC} \mod q \in R_q^{l \times m}$ and $\mathbf{W} = \mathbf{Y}_2 - \mathbf{DFC} \mod q \in R_q^{k \times m}$. A suitable rejection sampling condition need to be applyed before the output of the signature $\sigma = (r, \mathbf{Z}, \mathbf{W})$.

In a signature, the zero-knowledge property ensures that the signing process does not reveal any information about the secret key associated to the public key used in the verification process. The variant of EagleSign-V1 does not have the zero

knowledge property as pointed out by Tibouchi and Pulles in the PQC Forum of NIST (June 2023). Note that an attack similar to those of Tibouchi can be found in [1]. Our team have proposed EagleSign-V2 in December 2023 but Tibouchi (in pqc forum) remarks that "the deal $\mathbf{G}R_q$ can be easily recovered as the sum of the ideals $\mathbf{Z}R_q$ for the values $\mathbf{Z}$ in a handful of signature" and therefore $\mathbf{G}$ can be recovered. The attacks of Pells and Tibouchi are summarized in the IACR paper [81].
We propose in this work a third particular case where we avoid the above property of the ideal $\mathbf{G}R_q$ and the flaw in the old subsection 3.2 for zero knowledge property in EagleSign-V1 (NIST, June 2023). Therefore, the new scheme EagleSignV3 holds the zero knowledge property by using correctly the rejection sampling method of Lyubashevsky 2012 for secure signature over lattices. Notice that EagleSignV3 simplify the above longterm [resp: ephemeral] public key to $\mathbf{T} = (\mathbf{A} + \mathbf{D})\mathbf{G}^{-1}$ [resp: $\mathbf{p} := \mathbf{Ty} \mod q$] and uses $\mathbf{z} := \mathbf{y} + \mathbf{Gc}$ as a signature on a message M where $\mathbf{p}' = \text{HighBits}_q(\mathbf{p}, 2\gamma_2)$, $r := G(\mathbf{p}')$ and $\mathbf{c} \in B_\tau^l := H(M, r)$ (G and H are hash functions and H is a random oracle). A correct rejection sampling condition is applied before the output of the signature $(r, \mathbf{z})$.

1.3 Comparison with existing lattices based signatures

Since our signatures are lattices based signatures, we made only comparison with Dilithium that is standardized by NIST.

Table 1. Sizes and security levels for EagleSignV3, Dilithium

Dilithium :

NIST security level	2	3	5	5+
Module=M,	M	M	M	M
(Secret key recovery,Strong Unforgeable)				
BKZ block-size (b, b) to break (LWE,MSIS)	$(423, 423)$	$(624, 638)$	$(863, 909)$	$(1020, 1055)$
Best Known Classical bit-cost (LWE,MSIS)	$(123, 123)$	$(182, 186)$	$(265, 252)$	$(308, 298)$
Public key $\|pk\|$ in bytes	1312	1952	2592	2912
Signature $\|Sig\|$ in bytes	2420	3293	4595	5446
$\|Sig\| + \|pk\|$ in bytes	3732	5245	7187	8158

EagleSignV3:

NIST security level	2	5	2	5
Module=M, Ring=R	M	M	R	R
(Secret key recovery,Strong Unforgeable)				
BKZ block-size (b, b) to break (LWE,MSIS)	$(589, 402)$	$(1208, 883)$	$(589, 402)$	$(1208, 883)$
Best Known Classical bit-cost (LWE,MSIS)	$(172, 117)$	$(353, 258)$	$(172, 117)$	$(353, 258)$
Public key $\|pk\|$ in bytes	4128	9248	2080	4640
Signature $\|Sig\|$ in bytes	1824	4128	1824	4128
$\|Sig\| + \|pk\|$ in bytes	5952	13376	3904	8768

Notice that we can reduce the size of EagleSignV3 by working on the cyclotomic ring $R_q = \dfrac{\mathbb{Z}_q(X)}{(X^n - X^{n/2} + 1)}$ (of Lyubashevsky *et al* [31]) with $n = 972, 1296, 1536$ and $q = 1 \mod 3$ where NTT can also be used. It is possible also to implement EagleSignV3 over the field of NTRU-Prime $R_q = \dfrac{\mathbb{Z}_q(X)}{(X^n - X - 1)}$ with $X^n - X - 1$ irreducible, in order to avoid attacks targeting cyclotomic ring, but we loose NTT.

Organization of the paper: This paper is organized as follows.

- In Section 2, we recall some useful notions and we define basic operations and maps.
- In Section 3, we propose the specification of EagleSignV3.
- In Section 4, we adapt well known Fiat-Shamir techniques for security in QROM to EagleSignV3, EagleSignC and EagleSignD.
- And finally, in Section 5, we summarize the limitations and advantages of EagleSignV3.
- In appendix A, we explain at high level how the reference and optimized implementations were done for EagleSignV3.

2 Preliminaries

2.1 Notations and basic operations

In this subsection we use the same notations than Falcon, Bliss and Dilithium.

- The underlying rings of our signatures are $R = \dfrac{\mathbb{Z}(X)}{(X^n + 1)}$ and $R_q = \dfrac{\mathbb{Z}_q(X)}{(X^n + 1)}$.

- Regular font letters denote polynomials in R or R_q or elements in $\mathbb{Z}$ and $\mathbb{Z}_q$, bold lower-case letters represent column vectors of length l in in R^l or R_q^l and bold upper-case letters are matrices in $R^{k\times l}$ or $R_q^{k\times l}$ thus for $v, \mathbf{v}, \mathbf{V}$ the notation says that v is a scalar or a polynomial, $\mathbf{v}$ is a vector, and $\mathbf{V}$ is a matrix. For a vector $\mathbf{v}$ (resp: matrix $\mathbf{V}$).
- For an odd positive integer p , we define $r = z \mod^{\pm} p$, the centred reduction modulo p, to be the unique element r in the range $\frac{p-1}{2} \leq r \leq \frac{p-1}{2}$ such that $r \cong z \mod^{\pm} p$. We consider that $\mathbb{Z}_p = \{-\frac{p-1}{2}, \ldots, -1, 0, 1, \ldots, \frac{p-1}{2}\}$.
- For an even positive integer p , we define $r = z \mod^{\pm} p$, the centred reduction modulo p, to be the unique element r in the range $\frac{p}{2} < r \leq \frac{p}{2}$ such that $r \cong z \mod^{\pm} p$. We consider that $\mathbb{Z}_p = \{-\frac{p}{2}+1, \ldots, -1, 0, 1, \ldots, \frac{p}{2}\}$.
- We denote $r = z \mod^{\pm} p = z \mod^{\pm} p$ to simplify the notation throughout equations.
- For $f = \sum_{i=0}^{i=n-1} f_i X^i \in R_q, f_i \in \mathbb{Z}_p$, we denote $|f|_\infty = \max_i |f_i|$ and $|f|_1 = \sum_{i=0}^{i=n-1} |f_i|$.
- We have $|fg|_\infty \leq |f|_1 |g|_\infty$ in $R_q = \dfrac{\mathbb{Z}_q(X)}{(X^n+1)}$.
- S_η is the set of small polynomials which means that the element of S_η are polynomials with coefficients are in the interval $[-\eta, +\eta]$ and $B_{p,\tau} = \Big\{ f \in R_q / f = \sum_{i=0}^{i=n-1} f_i X^i, f_i \in \{-\dfrac{p-1}{2}, -1, 0, 1, \dfrac{p-1}{2}\}$ for $i = 0, \ldots, n-1$ with $\text{hwt}(f) = \tau \Big\}$ is the ball p-ary sparse polynomials. The entropy of $B_{p,\tau}$ is $\log \# B_{p,\tau}$ where $\# B_{p,\tau} = (p-1)^\tau \binom{n/2}{\tau}$. Notice that here p is odd.
  Notice that the elements of $B_{p,\tau}$ and $B_{p,\tau}$ have disjoint supports.

  The value of τ will be chosen such that the entropy of $B_\tau, B_{p,\tau}, B_{p,\tau}$ is greater than the security level.
- For $\mathbf{v} = (v_0, \ldots, v_{k-1})^T \in R_q^k$, we denote $|\mathbf{v}|_\infty = \max_i |v_i|_\infty$.
- The coefficients of the polynomials in $\mathcal{R}_q$ are in $\mathbb{Z}_p$.

2.2 Signature and its security model

A Randomized (deterministic) signature scheme consists of a triplet of polynomial-time algorithms (Genkey, Sig, Ver).

1. **Key Generation (Genkey)**: with input a security parameter K the key generation algorithm outputs a keypair (PK, SK) where PK, SK are related to each other throughout a hard mathematical problem (HMP).
2. **Signature algorithm (Sig)**:
   - Sig takes the security parameter K as input and produces a random r (skip in case of deterministic signature);
   - With input (SK, m, r) the signing algorithm Sig produces a signature σ.
3. **Verification (Ver)**: With input (m, σ, PK) the verification algorithm returns 1 if the signature is valid and 0 otherwise.

Security : When designing a signature scheme, we need to have in mind the following 4 fundamentals properties:

- (1) the signer should be able to make the verifier accept the proof if he really knows the secret key corresponding to the public key.
- (2) if the protocol succeeds (Ver outputs 1), then the verifier is convinced that the signer knows the secret key corresponding to the public key.
- (3) the verifier does not learn any information about the secret itself even if he sees many signatures (Zero-knowledge property).
- (4) nobody can forge a signature (which means that nobody is able to produce a valid signature without knowing the secret key)

Goldwasser, Micali and Rivest (in 1988) in [40], introduce the basic security notion for signatures called "existential unforgeability with respect to adaptive chosen- message attacks".

sEUF-CMA: Strong Unforgeability against Adaptive Chosen Message Attacks

For this, a reduction algorithm $\mathcal{R}$ and an attacker $\mathcal{A}$, simulate a the following game.

1. **Key generation**: $\mathcal{R}$ runs the algorithm Genkey with a security parameter K as input, to obtain the public key PK and the secret key SK, and gives PK to the attacker $\mathcal{A}$.
2. **The Queries of the adversary**: $\mathcal{A}$ may request a signature on any message $m \in \mathcal{M}$ (multiple adaptive requests of the message are allowed) and $\mathcal{R}$ will respond with (m, σ), without using the secret key but where $Ver(PK, m, \sigma) = 1$. The signatures already outputted by the oracle signature to the queries of the $\mathcal{A}$ are stored in a list $List(\mathcal{S})$.
3. **Strong forgery**: Eventually, $\mathcal{A}$ will output a pair (m, σ) and is said to win the game if $Ver(PK, m, \sigma) = 1$ and if $(m, \sigma) \notin List(\mathcal{S})$ (this last condition force the attacker $\mathcal{A}$ to output his own forgery (note that in this case of strong unforgeable it is allowed to the adversary to output $(m'', \sigma'') \notin List(\mathcal{S})$ assuming that $List(\mathcal{S})$ contains already signatures of the form (m'', σ''') with $\sigma''' \neq \sigma''$.

   The probability that $\mathcal{A}$ wins in the above game is denoted $Adv\mathcal{A}$.

A signature scheme (Genkey; Sig;Ver) is strongly existentially unforgeable with respect to adaptive chosen message attacks if for all probabilistic polynomial time attacker $\mathcal{A}$, $Adv\mathcal{A}$ is negligible in the security parameter K.

2.3 Hard problems over lattices

Definition 1 *(LWE). The learning with errors problem*
Consider the following equations $b_i = \boldsymbol{a}_i \boldsymbol{s}^t + e_i \mod q$ *for* $1 \leq i \leq k$ *where the* $\boldsymbol{a}_i, \boldsymbol{s} \in \mathbb{Z}_q^n$ *are chosen uniformly at random and the* e_i *(called the errors) are drawn from error distribution* χ*.*

– *Computational LWE: Given samples* $(\boldsymbol{a}_i, b_i)_i$ *compute* $\boldsymbol{s}$
– *Decisional LWE: Given samples* $(\boldsymbol{a}_i, b_i)_i$*, distinguish them from random samples in* $\mathbb{Z}_q^n \times \mathbb{Z}_q$

The decisional and the computational LWE problems are equivalent see Regev [68]. Moreover, if the secret key is selected from the distribution χ, then the LWE problem remains hard, see Applebaum *et al* [3].

The generalization of NTRU problem to matrix is the following.

Definition 2 *MNTRU: Module NTRU problem*

Consider $\mathbf{T} := \mathbf{D}\mathbf{G}^{-1} \mod q$ *where* $(\mathbf{D}, \mathbf{G})$ *are drawn independently from a distribution* χ*) (with* $\mathbf{G}$ *invertible).*

– *Computational MNTRU : Given samples* $\mathbf{T}$*, compute a valid* $sk = (\mathbf{D}, \mathbf{G})$*).*
– *Decisional MNTRU: Given samples* $\mathbf{T}$*, distinguish them from random samples in* $R_q^{k\times l} \times R_q^{k\times l}$*.*

In 2011, Damien Stehle and Ron Steinfeld [78] prove that the public key h of NTRU ($h = f^{-1}g$ for small f, g in R_q) is uniformly distributed when the secret f and the error g are chosen from a Gaussian distribution with large standard deviation. This result was generalized to MNTRU by Chuengsatiansup, Prest, Stehlé, Wallet and Xagawa in ModFalcon (ASIACCS 2020 [29]

Definition 3 *Module learning with errors problem:* MLWE
Let χ_s *and* χ_e *be two distributions over* R_q*.*
Consider $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e} \in R_q^k$ *where* $\mathbf{A}$ *is generted uniformly at random in* $R_q^{k\times l}$*, and* $\mathbf{s}, \mathbf{e}$ *are drawn independently from a distribution* χ

– *Computational MLWE problem: Given samples* $(\mathbf{A}, \mathbf{t})$ *compute* $\mathbf{s}$
– *Decisional MLWE problem: Given samples* $(\mathbf{A}, \mathbf{t})$*, distinguish them from random samples in* $R_q^{k\times l} \times R_q^k$

The case $k = l = 1$ is called $\mathrm{RLWE}_{q,n,\chi_s,\chi_e}$. We obtain the LEW problem when the above formula are viewed over $\mathbb{Z}_q$ rather than R_q.

Definition 4 *(l_∞-SIS). The short integer solution (Homogenus/Inhomogenus) problem*
Consider the following equation $\boldsymbol{t} = \boldsymbol{s}\boldsymbol{B} \mod q$ *where* $\boldsymbol{B} \in \mathbb{Z}_q^{n\times m}$*,* $m \geq n+1$ *is chosen uniformly at random and* $\boldsymbol{s} \in \mathbb{Z}_q^n$ *(called short vector) verify the upper bound* $|\boldsymbol{s}|_\infty \leq \beta \leq q-1$ *for some* $\beta \in \mathbb{R}$*.*

Computational l_∞*-*$SIS_{q,n,m,\beta}$*: Given* $(\boldsymbol{t}, \boldsymbol{B})$*, compute an appropriate* $\boldsymbol{s}$*.*

3 EagleSignV3; Design, security and parameters

In this section, we give the description of the two variants of our signature.

3.1 Basic functions

In this paper, we used the following functions of Dilithium for EagleSignV3:

Algorithm 1 Decompose

Require: $r \in \mathbb{Z}_q$

1: $r_0 = r \mod^{\pm} 2\gamma_2$
2: $r_1 = \dfrac{r - r_0}{2\gamma_2}$
3: **return** (r_0, r_1)

Algorithm 2 HighBits_q

Require: $r \in \mathbb{Z}_q$

1: $(r_0, r_1) = \text{Decompose}(r, 2\gamma_2)$
2: **return** r_1

Algorithm 3 LowBits_q

Require: $r \in \mathbb{Z}_q$

1: $(r_0, r_1) = \text{Decompose}(r, 2\gamma_2)$
2: **return** r_0

Lemma 1. *Let* $\mathbf{r}, \mathbf{s}$ *be vectors of elements in* R_q*. If* $|\mathbf{s}|_\infty \leq \beta$ *and* $|\text{LowBits}(\mathbf{r}, 2\gamma_2)|_\infty < \gamma_2 - \beta$*, then* $\text{HighBits}(\mathbf{r}, 2\gamma_2) = \text{HighBits}(\mathbf{r} + \mathbf{s}, 2\gamma_2)$

3.2 Description of EagleSign (General case)

The general case presented here is a slight modification of EagleSign and can be summarized at high level as follows.

1. Public and private keys:
   Keygen : it takes the security level and a system of parameters as inputs
   - $\mathbf{A} \in R_q^{k \times l}$ is a public matrix generated uniformly at random
   - $(\mathbf{F}, \mathbf{G}) \in S_{\eta_F}^{l \times l} \times S_{\eta_G}^{l \times l}$ are secret invertible matrices of small polynomials (generated uniformly at random).
   - $\mathbf{D} \in S_{\eta_D}^{k \times l}$ is a secret matrix of small polynomials (generated uniformly at random).
   - $\mathbf{T} := (\mathbf{A}\mathbf{F}^{-1} + \mathbf{D})\mathbf{G}^{-1} \in R_q^{k \times l}$
   - $pk := (\mathbf{A}, \mathbf{T})$ is the (longterm) public key.
   - $sk := (\mathbf{F}, \mathbf{G}, \mathbf{D})$ is the (longterm) private key (note that $\mathbf{F}, \mathbf{G}$ can be polynomial).
   - Output (pk, sk)
2. Signature
   Sig$(\text{M}, sk = (\mathbf{F}, \mathbf{G}, \mathbf{D}))$
   - $(\mathbf{Y}_1, \mathbf{Y}_2) \in S_{\eta_{y_1}}^{l \times m} \times S_{\eta_{y_2}}^{k \times m})$ is the ephemeral secret key;
   - $\mathbf{P} := \mathbf{T}\mathbf{Y}_1 + \mathbf{Y}_2 \in R_q^{k \times m}$ is ephemeral public key (this formula is modified and the old formula of the signature EagleSign was $\mathbf{P} := \mathbf{T}\mathbf{Y}_1 + \mathbf{Y}_2 \mod q$);
   - $r := G(\mathbf{P})$;
   - $\mathbf{C} \in S_{\eta_c}^{l \times m} := H(M, r)$;
   - $\mathbf{Z} := \mathbf{Y}_1 + \mathbf{G}\mathbf{F}\mathbf{C} \mod q \in R_q^{l \times m}$ (this formula is modified and the old formula of the signature EagleSign was $\mathbf{Z} := \mathbf{G} - \mathbf{Y}_1 + \mathbf{F}\mathbf{C}) \mod q$);

- $\mathbf{W} := \mathbf{Y}_2 - \mathbf{DFC} \mod q \in R_q^{k\times m}$;
- Reject if some appropriate upper-bounds of the norms of $\mathbf{Z}, \mathbf{W}$ are not verified (suitable rejection sampling)
- Output the signature $\sigma = (r, \mathbf{Z}, \mathbf{W})$

3. Verification
   $\mathbf{Ver}(\sigma = (r, \mathbf{Z}, \mathbf{W}), pk = (\mathbf{A}, \mathbf{t}))$
   - $\mathbf{C} \in S_{\eta_c}^{l\times m} =: H(M, r)$
   - $\mathbf{V} = \mathbf{TZ} - \mathbf{AC} + \mathbf{W} \mod q$
   - Reject if some appropriate upper-bounds (provided by the designer) of the norms of $\mathbf{Z}, \mathbf{W}$ are not verified
   - Reject if $\mathbf{C} \neq H(M, G(\mathbf{V}))$
   - Otherwise accept

Tibouchi and Pulles in the PQC Forum of NIST (June 2023) have propose an attack that proves that EagleSign-V1 does not have the zero knowledge property and similarly Tibouchi have propose an attack on EagleSign-V2 in December 2023. A similar attack to those of can be found in [1] at page 24.

We have propose the following countermeasures:

- we choose a new particular case of the public key ($F = 1$) and and we change the formula of the signature for this particular instantiation
- we introduce rejection sampling of the Lyubashevsky for secure signature over lattices
- we change the simulation of the signature in security proof in ROM.

In the following, we propose the new variant EagleSignV3 that holds the zero knowledge property.

3.3 Description of EagleSignV3

In this subsection, we propose the three following detailed algorithms for our signature in case $m = 1, k, l \in \{1, 2, \ldots\}$. We use the following functions and notations:

1. The transformation MatrixUnifEtaPolyn maps a uniform seed $\rho \in \{0,1\}^{512}$ to a matrix $A \in \mathcal{R}_q^{k\times l}$ (for $k, l = 1, 2, \ldots$) in NTT domain representation;
2. The function GenUnifEtaPolyn, with input a seed, generates uniformly at random a polynomial in the set S_η.
   The functions MatrixUnifEtaPolyn and VectorUnifEtaPolyn call GenUnifEtaPolyn with different seed as input to generate each element of the matrix $\mathbf{D}$ in $S_\eta^{k\times l}$, the matrix $\mathbf{G}$ in $S_\eta^{l\times l}$ (invertible) and the vector $\mathbf{y}$ in $S_{\gamma_1}^l$.
3. The function CRH (resp. CRH1) is a collision resistant hash used in our signature scheme and mapping to $\{0,1\}^{768}$ (resp. $\{0,1\}^{512}$).
4. The function G is a multi-collision resistant hash used in our signature scheme and mapping to $\{0,1\}^{512}$.
5. $H : \{0,1\}^\star \to B_\tau^l$ is a cryptographic hash function used to generate $\mathbf{c} \in B_\tau^l$ which calls the function GenSparseSmallPolyn.

6. The function GenRandoms is interpreted as SHAKE-256 in our implementation.

7. We consider the following bounds to make sure that each output of the signature is short enough :
$\beta = l \times \eta \times \tau, \delta_B = \gamma_1 - \beta$, $\delta'_B = \gamma_2$ and $\alpha = \max(2\delta_B, 2\delta'_B) < (q-1)/2$.

Note that the description of these previous functions is given is the section A.4.

Algorithm 4 : EagleSignV3 Key generation algorithm

Require: the security parameter 1^{256}
1: $\beta \leftarrow \{0,1\}^{256}$;
2: $(\beta_1, \beta_2, \rho, \text{key}) := \text{GenRandoms}(\beta)$ ▷ $(\beta_1, \beta_2, \rho, \text{key}) \in (\{0,1\}^{256})^{3+1}$
3: $(\beta_1) = \text{Hash}(\beta_1)$, ▷ we use SHAKE-256 for Hash to renew β_1
4: $\mathbf{G} := \text{MatrixUnifEtaPolyn}(\beta_1, \eta, l, l)$ ▷ $\mathbf{G} \in S_\eta^{l \times l}$
5: **if** $\mathbf{G}$ is not invertible in $\mathcal{R}_q^{l \times l}$ **then**
6: Go to step (3);
7: **end if**
8: $\mathbf{D} := \text{MatrixUnifEtaPolyn}(\beta_2, \eta, k, l)$; ▷ $\mathbf{D} \in S_\eta^{k \times l}$,
9: $\mathbf{A} := \text{MatrixUnifEtaPolyn}(\rho)$; ▷ $\mathbf{A} \in \mathcal{R}_q^{k \times l}$
10: $\mathbf{T} := (\mathbf{A} + \mathbf{D})\mathbf{G}^{-1} \mod q$;
11: $\text{tr} := \text{CRH1}(\rho, \mathbf{T})$; ▷ $\text{tr} \in \{0,1\}^{512}$
12: $\text{sk} := (\rho, \text{tr}, (\mathbf{D}, \mathbf{G}), \text{key})$; ▷ the longterm private key
13: $\text{pk} := (\rho, \mathbf{T})$; ▷ the longterm public key
14: **return** (pk, sk)

Remark: The parameter $'\text{key}'$ is only used in case of deterministic signature.

EagleSignV3 **Signature algorithm**	**EagleSignV3** **Verification algorithm**
1. a message M, a secret key $\text{sk} = (\rho, \text{tr}, (\mathbf{D}, \mathbf{G}, \text{key})$ 2. $\mu \in \{0,1\}^{384} := \text{CRH}(\text{tr}, M)$	
3. $\lambda \leftarrow \{0,1\}^{256}$	1. signature $\sigma = (r, \mathbf{z})$, public key
4. $\mathbf{z} := \perp$	$(\rho, \mathbf{T})$ and parameters $\beta, \gamma_1, \gamma_2$
5. $\mathcal{K} := 0$	2. $\text{tr} \in \{0,1\}^{384} := \text{CRH1}(\rho, \mathbf{t})$
6. While $\mathbf{z} := \perp$	3. $\mu \in \{0,1\}^{384} := \text{CRH}(\text{tr}, M)$
6.1 $\mathbf{y} \leftarrow S^l_{\gamma_1} := \text{VectorUnifEtaPoly}(\lambda, \mathcal{K})$	4. $\mathbf{c}' \in B^l_\tau := H(\mu, r)$
6.2 $\mathbf{p} := \mathbf{Ty} \mod q \in R^k_q$	5. $\mathbf{A} \leftarrow \mathcal{R}^{k \times l}_q := \text{MatrixUnifEtaPoly}(\rho)$
6.3 $\mathbf{p}' = \text{HighBits}_q(\mathbf{p}, 2\gamma_2)$	6. $\mathbf{v} := \mathbf{Tz} - \mathbf{Ac}' \mod q$
6.4 $r := G(\mathbf{p}')$	7. $\mathbf{v}' := \text{HighBits}_q(\mathbf{v}, 2\gamma_2)$
6.5 $\mathbf{c} \in B^l_\tau := H(\mu, r)$	8. $r' = G(\mathbf{v}')$
6.6 $\mathbf{z} := \mathbf{y} + \mathbf{Gc} \mod q$	9. If $\lvert\mathbf{z}\rvert_\infty \geq \gamma_1 - \beta$ or $\mathbf{c}' \neq H(\mu, r'))$
6.7 $\mathbf{r}_0 := \text{LowBits}_q(\mathbf{p} + \mathbf{Dc}, 2\gamma_2)$	10.1 return 0
6.8 If $\lvert\mathbf{z}\rvert_\infty \geq \gamma_1 - \beta$	11. Else
or $\lvert\mathbf{r}_0\rvert_\infty \geq \gamma_2 - \beta$ then $\mathbf{z} := \perp$	11. Return 1
6.9 $\mathcal{K} := \mathcal{K} + l$	
7. End While	
8. Return $\sigma := (r, \mathbf{z})$ as signature	

Remark:
– Notice that $\mathbf{G}$ (bold case) and G are different since $\mathbf{G}$ is a matrix and G is a hash function.
– In case of probabilistic signature λ is a random and in case of deterministic signature $\lambda = (\mu,' \text{key}')$
– We introduce in the previous algorithm a rejection sampling to make sure that $\mathbf{z}$ has the zero knowledge property which means that the collection of many signature $\mathbf{z}$ don't leak any information about the secret key.
– Validity of the signature (optional): to defeat fault signature attacks, we can compute r_0 as $\mathbf{r}_0 := \text{LowBits}_q(\mathbf{Tz} - \mathbf{Ac}, 2\gamma_2)$ instead of $\mathbf{r}_0 := \text{LowBits}_q(\mathbf{p} + \mathbf{Dc}, 2\gamma_2)$.

Correctness of the signature: Easy to verify.

3.4 Comparison of the design EagleSignV3 and the two old EagleSign

Tibouchi *et al* have propose an attack on EagleSign in the NIST forum for pq-signqtures. This attack shows that EagleSign-V1 (and its successor EagleSign-V2 in December 2023) don't have the zero knowledge property which means that the signature can leak the private key.
We propose the following countermeasures to this attack (and hence reach the zero knowledge property):

- we choose a new variant the signature;
- we apply the Lyubashevsky rejection sampling method for secure signature over lattices;
- consequently, we adapt the simulation of the signature in security proof in ROM,
- and we choose new parameters for $q, n, \ldots$.

For a comparison between the two old EagleSign (from EagleSign submitted to NIST competition in June 2023 and to NIST pqc forum in Decembre 2023) and EagleSignV3, we have made the following changes summarized in the following table.

Table 2. Summary the two old variant of EagleSign
Public Key=pk, secret key =sk, Ephemeral public key=Epk, Ephemeral secret key=Esk, Signature=Sig, Boubs=Bs, Rejection sampling of Lyubashevsky=RS

	Old EagleSign	EagleSign-V1	EagleSign-V2
pk	$pk := (\mathbf{A}, \mathbf{T} := (\mathbf{AF}^{-1} + \mathbf{D})\mathbf{G}^{-1})$	$\mathbf{T} := (\mathbf{A} + \mathbf{D})\mathbf{G}^{-1}$	$\mathbf{T} := (\mathbf{AF}^{-1} + \mathbf{D})g^{-1}$
sk	$sk := (\mathbf{F}, \mathbf{G}, \mathbf{D})$	$(\mathbf{G}, \mathbf{D})$	$(\mathbf{F}, g, \mathbf{D})$
Epk	$\mathbf{P} := \mathbf{AF}^{-1}\mathbf{Y}_1 + \mathbf{Y}_2$	$\mathbf{p} := \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$;	$\mathbf{p} := (\mathbf{AF}^{-1} + \mathbf{D})\mathbf{y}$
Esk	$(\mathbf{Y}_1, \mathbf{Y}_2)$	$(\mathbf{y}_1, \mathbf{y}_2)$;	$\mathbf{y}$
Sig	$\mathbf{Z} := \mathbf{GU} \mod q$ and $\mathbf{W} := \mathbf{Y}_2 - \mathbf{DU} \mod q$, $\mathbf{U} := \mathbf{Y}_1 + \mathbf{FC} \mod q$;	$\mathbf{Z} := \mathbf{Gu} \mod q$ $\mathbf{w} := \mathbf{y}_2 - \mathbf{Du} \mod q$ $\mathbf{u} := \mathbf{y}_1 + \mathbf{c} \mod q$	$\mathbf{Z} := g\mathbf{u} \mod q$ None $\mathbf{u} := \mathbf{y} + \mathbf{Fc} \mod q$
Bs	None	None	$\mathbf{r}_0 := \text{LowBits}_q(X, 2\gamma_2)$; $\mathbf{X} = \mathbf{p} + \mathbf{DFc}$; $\|\mathbf{z}\|_\infty \geq t_g(\gamma_1 - \beta)$ $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - lt_D\beta$ $\|\mathbf{X}\|_\infty \geq \frac{q-1}{2} - lt_D\beta$
RS	None	None	Yes

Table 3. Summary the new EagleSign
Public Key=pk, secret key =sk, Ephemeral public key=Epk, Ephemeral secret key=Esk, Signature=Sig, Boubs=Bs, Rejection sampling of Lyubashevsky=RS

	New variant of EagleSign	EagleSignV3: new instantiation
pk	$pk := (\mathbf{A}, \mathbf{T} := (\mathbf{AF}^{-1} + \mathbf{D})\mathbf{G}^{-1})$	$\mathbf{T} := (\mathbf{A} + \mathbf{D})\mathbf{G}^{-1}$
sk	$sk := (\mathbf{F}, \mathbf{G}, \mathbf{D})$	$(\mathbf{G}, \mathbf{D})$
Epk	$\mathbf{P} := \mathbf{TY}_1 + \mathbf{Y}_2$	$\mathbf{p} := \mathbf{Ty}$
Esk	$(\mathbf{Y}_1, \mathbf{Y}_2)$	$\mathbf{y}$;
Sig	and $\mathbf{W} := \mathbf{Y}_2 - \mathbf{DFC} \mod q$, $\mathbf{Z} := \mathbf{Y}_1 + \mathbf{GFC} \mod q$;	 $\mathbf{z} := \mathbf{y} + \mathbf{Gc} \mod q$
Bs	yes	$\beta = \|\mathbf{Gc}\|_\infty = l\eta(\tau_1 + \tau_2)$ $\mathbf{X} = \mathbf{Tz} - \mathbf{Ac}$; $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ $\mathbf{r}_0 := \text{LowBits}_q(X, 2\gamma_2)$ $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$
RS	yes	yes

3.5 Security analysis of EagleSignV3

3.5.1 Choosing the modulus and the number of repetitions In our signature EagleSignV3, we need to satisfy the following conditions (otherwise we repeat):
and $\mathbf{p} := \mathbf{Ty} \mod q$;
(1): $|\mathbf{r}_0|_\infty < \gamma_2 - \beta$ with $\mathbf{r}_0 := \text{LowBits}_q(\mathbf{p} + \mathbf{Dc}, 2\gamma_2)$;
(2): $|\mathbf{z}|_\infty < \gamma_1 - \beta_G$ with $\mathbf{z} := \mathbf{y} + \mathbf{Gc}$.
As in Dilithium, the probability for (1) is $\big(\frac{2(\gamma_2 - \beta_D) - 1}{2\gamma_2 - 1}\big)^{nk} \sim e^{-nk\beta/\gamma_2}$ and those of (2) is $\big(\frac{2(\gamma_1 - \beta) - 1}{2\gamma_1 - 1}\big)^{nl}$.

Assuming that γ_1 and γ_1 are big enough and that the two relations are independent, we see that the the probability of both (1) and (2) is $e^{-nk\beta/\gamma_2}.e^{-nl\beta/\gamma_1} \sim e^{-n(l\beta/\gamma_1 + k\beta/\gamma_2)}$, thus the number of repetitions is the inverse which is $N = e^{n(l\beta/\gamma_1 + k\beta/\gamma_2)}$. We choose $\gamma_1 \geq 2^{\lceil \log_2(nl\beta) \rceil}$ and $\gamma_2 \geq 2^{\lceil \log_2(nk\beta) \rceil}$ such that the number of repetitions verify $1 \leq N \leq 9$.

We choose q such that q is prime and $q > 4\max(\gamma_1, \gamma_2)$ with additional conditions depending on the R_q and the kind of NTT.

3.5.2 Zeroknowledge proof for EagleSignV3 We obtain the zero knowledge property for EagleSignV3 by using the rejection sampling of Lyubashevsky [54].
Tibouchi have pointed out a flaw in the previous proof in EagleSign-V1 qnd EagleSign-V2. We solve the problem with the new formula introduced for the signature $\mathbf{z} = \mathbf{y} + \mathbf{Gc}$ (and $\mathbf{F} = 1$) instead of $\mathbf{z} = \mathbf{G}(\mathbf{y}_1 + \mathbf{c})$ for EagleSign-V1 or $\mathbf{z} = g(\mathbf{y} + \mathbf{Fc})$ for EagleSign-V2.

Let us compute the probability of $(r, \mathbf{z})$ outputted by our signature taken over the randomness of $\mathbf{y}$ and the random oracle H which is modelled as a random function with $\mathbf{c} = H(\mu, r)$, $\mathbf{z} = \mathbf{y} + \mathbf{Gc}$) and $|\mathbf{Gc}|_\infty < \beta = l\eta(\tau_1 + \tau_2)$. Similarly to Dilithium, we have the following.
$\mathbb{P}(r, \mathbf{z}) = \mathbb{P}(r) \times \mathbb{P}(\mathbf{y} = \mathbf{z} - \mathbf{G}H(\mu, r)|r)$
Whenever $|\mathbf{z}|_\infty < \gamma_1 - \beta$, then the above probability is exactly the same for every such tuple $(r, \mathbf{z})$. This is because $|\mathbf{Gc}|_\infty \leq \beta$, and thus $|\mathbf{y}| = |\mathbf{z} - \mathbf{Gc}|_\infty < \gamma_1$, which is a valid value of $\mathbf{y}$. Therefore, by rejection sampling, if we only output $\mathbf{z}$ where $|\mathbf{z}|_\infty < \gamma_1 - \beta$ (by the rejection sampling of Lyubashevsky 2012), then the resulting distribution of $(r, \mathbf{z})$ will be uniformly random over $\{0,1\}^{|r|} \times S^l_{\gamma_1 - \beta - 1}$. Consequently, we can use the previous result to simulate the signature in the security proof (see next subsubsection).

3.5.3 Security proof in ROM for EagleSignV3 In this subsection, we adapt to lattices, the tools, techniques and frameworks for security proof developed by Pointcheval *et al.* [67] for Elgamal-like signatures (DSA, KCDSA, Schnorr,...) where the underlying hard problem was the discrete logarithm problem. To design a security proof in ROM for EagleSignV3, we use the following steps.

1. Protection against secret key recovery: we need to prove that recovering the private key from the public key is equivalent to solving hard instance in a specified lattice problem namely the MLWE problem in our case.

2. Simulation of the random oracle H: the cryptographic hash function H of the signature is considered to be an ideal random function that the attacker can query as an oracle. For each new query of the attacker, the simulator chooses uniformly at random a value in the output set of the real hash function and sends it as response. This answer needs to be independent from previous query/response pairs stored in a data base L_H by the simulator. If a query is replayed by the attacker, the simulator finds the correct answer in L_H.
3. Simulation of the signature: without the private key and by controlling the ideal hash function H, the simulator design a signature algorithm able to produce valid signatures in polynomial time with a hight probability.

   In our simulation, as proved by Pointcheval *et al* [67] for classical DSA, it is not necessary to consider the second hash function G as a random oracle thus the use of random oracles is minimizing. G will be just considered as a multi-collision-resistance function: G is said j-collision-resistant, if it is hard to find $(u_1, \ldots, u_j)$ pairwise distinct elements such that $G(u_1) = \ldots = G(u_j)$.

4. Signature forgery: Using an adaptively chosen-message attack against the legitimated signer, the attacker produces a valid signature forgery with Q_H queries to the ideal hash function H and Q_S queries to the oracle signature. For each new query to H, L_H is updated with the corresponding query/response pair. To be a real attack, it is assumed the valid signature of the attacker has not been sent as a call to the signature oracle.
5. Solving a MSIS problem using signature forgery (with he following steps):
   - Since the attacker don't control the ideal random function H, from a signature forgery of the attacker, the "forking lemma" is used to show that, she can construct two signatures with the same fixed values $(M/\mu, r)$ but H produce different responses $\mathbf{c}$ and $\mathbf{c}'$ (which really means that different ideal random functions are used; this scenario is possible since the attacker don't control H in ROM).
   - The previous scenario produces collusions throughout G from the positive answer of the verification process.
   - Two valid forged signatures (with collusion) are used to show how to compute a short non-zero vector as a solution of a MSIS problem with l_∞ norm.

Theorem 1. *Assume that an attacker $\mathcal{A}$ produce an existential forgery of the Eagl Sign after Q_H calls to H and Q_S calls to the simulator for signature, under an adaptively chosen message attack with probability ϵ, then by forking lemma, the MSIS-l_∞ problem $(\mathbf{T}|\mathbf{A}|\mathbf{I}_k)\mathbf{x}^T = 0$ can be solved in polynomial time for $||\mathbf{x}||_\infty \leq \alpha = \max(2\delta_B, 2\delta'_B) < \frac{q-1}{2}$ where $\mathbf{T}$ is the public key of EagleSignV3, $\delta_B = \gamma_1 - \beta$ and $\delta'_B = \gamma_2$. Note that the probabilities are taken over random tapes, random oracles, messages and public/private keys (longterm and ephemeral).*

Proof. A) Protection against secret key forgery:
The longterm public key $\mathbf{T} = (\mathbf{A} + \mathbf{D})\mathbf{G}^{-1} \mod q$ can be converted in LWE instances where $\mathbf{D}$ is the error and $\mathbf{G}$ is the secret . Therefore the public key is indistinguishable from random whenever the corresponding instance of LWE is indistinguishable from random, reason why we choose the coefficients of the polynomial in $\mathbf{A}$ uniformly at random in R_q and the security level concerning the public key is estimated via the classic LWE estimator.

B) Simulation of the signature:
We need to simulate the signature without the private key with the ideal hash function under control. From the result of the zero knowledge property, we know that the distribution of the signature $(r, \mathbf{z})$ is uniformly random over $\{0,1\}^{|r|} \times S^l_{\gamma_1-\beta-1}$. We will then use this fact in the following to simulate the signature.

1. input a message M;
2. generate randomly $\mathbf{z}$ such that $|\mathbf{z}|_\infty < \gamma_1 - \beta$;
3. $\text{tr} = \text{CRH1}(\rho, \mathbf{T})$;
4. $\mu = \text{CRH}(\text{tr}, M)$;
5. generate randomly $\mathbf{c} \in B^l_\tau$;
6. $\mathbf{v} = \mathbf{Tz} - \mathbf{Ac} \mod q$
7. $\mathbf{r}_0 := \text{LowBits}_q(\mathbf{v}, 2\gamma_2)$;
8. If $|\mathbf{r}_0|_\infty \geq \gamma_2 - \beta$
   then Go To (2)
9. compute $\mathbf{v}' = \text{HighBits}_q(\mathbf{v}, 2\gamma_2)$
10. compute $r = G(\mathbf{v}')$;
11. define $\mathbf{c} = H(\mu, r) \in B^l_\tau$ and update the data base L_H of the oracle hash function with the query/response $(M/\mu, r)/\mathbf{c}$;
12. Output the signature $(M, r, \mathbf{z})$.

The simulation of the signature is indistinguishable.

D) Forking for solving the MSIS problem :
If the attacker $\mathcal{A}$ output a valid signature $((M/\mu, r, \mathbf{c}), \mathbf{z})$ where c can be found in L_H with the prefix $(M/\mu, r)$ with probability ϵ for a new message M with less than Q_H calls to the hash function H, then by forking technique we obtain two valid signatures of the same message M and fixed values namely $(M/\mu, r, \mathbf{c}), \mathbf{z}$ and $(M/\mu, r', \mathbf{c}'), \mathbf{z}'$ with $r = r'$ and $\mathbf{c} \neq \mathbf{c}'$. From $r = r'$, we deduce $v = v'$ with a high probability, therefore $\text{HighBits}_q(\mathbf{Tz} - \mathbf{Ac} \mod q, 2\gamma_2) = \text{HighBits}_q(\mathbf{Tz}' - \mathbf{Ac}', 2\gamma_2)$. Thus $(\mathbf{T}(\mathbf{z}-\mathbf{z}') - \mathbf{A}(\mathbf{c}-\mathbf{c}')) \mod q = \mathbf{r}'_0 - \mathbf{r}_0 \mod q$ where $\mathbf{r}_0 := \text{LowBits}_q(\mathbf{Tz}-\mathbf{Ac}, 2\gamma_2)$ and $\mathbf{r}'_0 := \text{LowBits}_q(\mathbf{Tz}'-\mathbf{Ac}', 2\gamma_2)$. Hence, we have $(\mathbf{T}|\mathbf{A}|I_k))(\mathbf{z}-\mathbf{z}', \mathbf{c}'-\mathbf{c}, \mathbf{r}_0-\mathbf{r}'_0)^T = 0$. Now, put $\mathbf{x} = (\mathbf{z} - \mathbf{z}', \mathbf{c} - \mathbf{c}', \mathbf{r}'_0 - \mathbf{r}_0)$, since $\mathbf{c}' - \mathbf{c} \neq 0$ and $||\mathbf{x}||_\infty \leq \alpha = \max(2\delta_B, 2\delta'_B)$ then we see that $\mathbf{x}$ is a nonzero short solution of the MSIS problem.

NB: Note that we don't need to use the SelfTargetMSIS problem and the Hint vector of Dilithium in our security proof.

3.6 Sizes and security levels for EagleSignV3

Sizes for EagleSignV3
The sizes of the Public key and the Signature are the following. The signature and the public key of EagleSignV3 are respectively $\sigma = (r, \mathbf{z})$ and $pk = (\rho, \mathbf{T})$ where $\mathbf{z} = \mathbf{y} + \mathbf{Gc} \mod q, \mathbf{c} \in B^l_\tau, \mathbf{D} \in S^{k\times l}_{\eta_D}, \mathbf{G} \in S^{l\times l}_{\eta_G}$ with $|\mathbf{z}|_\infty \leq \gamma_1 - \beta < \gamma_1$, and $\mathbf{T} = (\mathbf{A} + \mathbf{D})\mathbf{G}^{-1} \mod q \in R_q$, then $|\sigma| = \lceil 32 + n \times (l \times \log_2(2 \times \gamma_1))/8 \rceil$ bytes and $|pk| = \lceil 32 + n \times (k \times l \times \log_2(q))/8 \rceil$ bytes.

Security levels for EagleSignV3
For a complete study of the estimation of the security level of LWE and NTRU -like schemes proposed at NIST, one can see the recent work of Albrecht, Curtis, Deo,

Davidson, Player, Postlethwaite, Virdia, Wunderer in [4] : Estimate all the LWE and NTRU schemes (PQC-Forum January 2018). In their paper [4], the authors point out the sources of divergence (instantiation of the SVP oracle in BKZ by sieving method or enumeration method, treatment of polynomial factor) in estimated security level of the ideal lattice-based schemes proposed to NIST.
Many techniques for improving lattice-based cryptanalysis where proposed recently [2, 8, 10, 28, 30, 46, 47, 56, 63, 73–75, 82, 86]. Moreover, vulnerabilities in ideal lattice-based schemes where pointed out by many authors [10, 16, 24, 30]. Based on these results, some authors claim that the security of lattice-based cryptography over the rings is not well understood (see Bernstein *et al.* in NTRU LPRime [59]). Nevertheless, currently, as far as we know, these algebraic structure does not figure into the cost of the best known attacks on NTRU-RLWE-like schemes and in general, no algorithm is known that can exploit enough the ring structure and that is thus working more efficiently on ideal-lattices than classical lattices [2, 9, 16, 59]. Therefore, we can analyse the hardness of our signature over standard lattices.
For recent advances on the security analysis of lattices, we have the interesting work of E. Wenger, E. Saxena, M. Malhou, E. Thieu and K. Lauter in *Benchmarking Attacks on Learning with Errors* 2024 [84].
For advances and background for solving uSVP and similar problems, we refer to [2, 4, 5, 7, 8, 12, 84]. Recall that BKZ lattice reduction algorithm (which is a blockwise variant of the LLL algorithm) proceeds by sublattice reduction using a SVP oracle in a smaller dimension b. With BKZ, the best known classical algorithm (respectively: quantum sieving algorithm) [2, 12, 28, 46] for the primal/dual attack [2, 8, 19] with block size b of MLWE or MNTRU-like schemes, have costs of $2^{0.292b}$ (respectively: $2^{0.265b}$ with Grover speedups [41]). Therefore, currently (June in 2023, as far as we know), we must at least use $2^{0.265b}$ (or the "paranoid" lower bound $2^{0.2075b}$ given in [2, 4]) to compute the security level.

To estimate the security level, we use the lattice estimator of Albrecht *et al.* [4–6] (lattice-estimator-main with Sagemath and python) to estimate the security of the longterm public key and the ephemeral public key. We use the tool of Crystal Dilithium to estimate the security of MSIS for unforgeability.
The following algorithms 3 are covered by the estimator that we have used in EagleSignV3 security: meet-in-the-middle exhaustive search, coded-BKW, dual-lattice attack and small/sparse secret variant, lattice-reduction and enumeration, primal attack via uSVP [19], Arora-Ge algorithm [16] using Gröbner bases.
To estimate the security level we consider that our public key $\mathbf{T} := (\mathbf{A} + \mathbf{D})\mathbf{G}^{-1}$ mod q follows the LEW model were $\mathbf{G}$ is the secret and $\mathbf{D}$ is the error) and we use the LWE estimator. To evaluate the security of MSIS, we use the tool of Dilithium for the unforgeability security analysis based on MSIS problem upper bounded by $\alpha = \max(2\delta_B, 2\delta'_B)$ with l_∞ norm.
Parameters Selection, Size and NIST Security Levels
We summarize in the following table the results of the LWE estimator and the tool of Dilithium for MSIS. The table contains also, the size of the public key and the signature for each security level.

Table 4. Parameters Selection, Size and NIST Security Levels

NIST security level	2	5	2	5
Variant	Module	Module	Ring	Ring
(k,l)	$(2,2)$	$(2,2)$	$(1,1)$	$(1,1)$
$n=2^s$	512	1024	1024	2048
$(q=1 \mod 2n)\ \ q=$	59393	249857	59393	249857
$\mathbf{c} \in B_\tau^l, \tau$	8	16	16	32
Entropy of $\mathbf{c} = l(\log_2(\binom{n}{\tau})) + \tau)$	128	262	131	265
$(\mathbf{G},\mathbf{D}) \in S_\eta^{l\times l} \times S_\eta^{l\times l} : \eta =$	1	1	1	1
$\beta = l.\eta.\tau,$	16	32	16	32
(γ_1,γ_2)	$(2^{14},2^{14})$	$(2^{16},2^{16})$	$(2^{14},2^{14})$	$(2^{16},2^{16})$
Number of Repetitions	7.38	7.38	7.38	7.38
Strong Unforgeable $\alpha = \max\{2.(\gamma_1-\beta), 2\gamma_2\}$				
BKZ block-size b to break LWE	402	883	402	883
Best Known Classical bit-cost	117	258	117	258
Best Known Quantum bit-cost	107	234	107	234
Longterm Secret key recovery $(\mathbf{G},\mathbf{D})$				
BKZ block-size b to break LWE	589	1208	589	1208
Best Known Classical bit-cost	172	353	172	353
Best Known Quantum bit-cost	156	272	156	272
Sizes in bytes				
Public key $\|pk\|$	4128	9248	2080	4640
Signature $\|Sig\|$	1824	4128	1824	4128
$\|Sig\| + \|pk\|$	5952	13376	3904	8768

As remarked above, over the the cyclotomic ring $R_q = \dfrac{\mathbb{Z}_q(X)}{(X^n - X^{n/2}+1)}$ of Lyubashevsky *et al* [31], we can reduce the size of ring based variant of EagleSignV3 by working with $n = 972, 1296, 1536$ and $q = 1 \mod 3$ and using an appropriate NTT . To avoid attacks targeting cyclotomic rings, we can instantiate EagleSignV3 over the field of NTRU-Prime $R_q = \dfrac{\mathbb{Z}_q(X)}{(X^n - X - 1)}$ with $X^n - X - 1$ irreducible. Nevertheless, in this later case, the size of the modulus will probably increase by 1 to 3 bits.

4 Security proof in QROM for EagleSignV3

Our signature is secure in ROM and is a Fiat-Shamir signature scheme with aborts, thus we can obtain the security in QROM (see [17,17,34]), and for the shake of completeness, a complete proof in QROM will be designed later.
Recently Barbosa *et al.* [17] and Devevey *et al.* [26], independently have detected some flaws in existing security proof for Fiat-Shamir signatures (such as Dilithium) and have proposed important improvements.
Note that the authors of Dilithium say the following: "In our opinion, evidence is certainly mounting that the distinction between signatures secure in the ROM and QROM will soon become treated in the same way as the distinction between schemes secure in the standard model and ROM – there will be some theoretical differences, but security in practice will be the same".

5 Advantages and Limitations for EagleSignV3

Advantages: The public key is a mix of MNTRU (Module NTRU) and MLWE (Module learning With Error). Many other variants can be investigated in the future. For recommended parameters, the signature size and public key size of ring variant of EagleSignV3 is similar to those of Dilithium but its public key is based on a different problem.
Limitations: It has the same limitations as any lattices based digital signature regarding the long term security.

.

References

1. Agrawal, S., Stehlé, D., & Yadav, A. Round-optimal lattice-based threshold signatures, revisited. Cryptology ePrint Archive 2022 *https://eprint.iacr.org/2022/634*.
2. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. *Post-quantum key exchange - A new hope*. In T. Holz and S. Savage, editors, Proceedings of the 25th USENIX Security Symposium, pages 327-343. USENIX Association, 2016. URL: https://www.usenix.org/conference/usenixsecurity16/techniqueal-sessions/presentation/alkim.
3. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. *Fast cryptographic primitives and circular-secure encryption based on hard learning problems*. In CRYPTO, pages 595-618, 2009
4. M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. Postlethwaite, F. Virdia, T. Wunderer, *Estimate all the LWE and NTRU schemes!* https://estimate-all-the-lwe- ntru-schemes.github.io/paper. pdf. NIST Call for transision to quantum-resistant cryptography (November 2017)
5. Martin Albrecht. *Security estimates for the learning with errors problem*, 2017. Version 2017-09-27, https://bitbucket.org/malb/lwe-estimator. 21
6. Martin R. Albrecht, Rachel Player and Sam Scott. *On the concrete hardness of Learning with Errors*. Journal of Mathematical Cryptology. Volume 9, Issue 3, Pages 169–203, ISSN (Online) 1862-2984, ISSN (Print) 1862-2976 DOI: 10.1515/jmc-2015-0016, October 2015
7. Martin Albrecht and Amit Deo. *Large modulus Ring-LWE ≥ Module-LWE*, 2017.To appear. https: //eprint.iacr.org/2017/612. 22
8. Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. *Revisiting the expected cost of solving uSVP and applications to LWE*. In Advances in Cryptology -ASIACRYPT 2017 -23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I, pages 297 322, 2017
9. M. R. Albrecht, C. Cid, J.C. Faugere, and L. Perret. *Algebraic algorithms for LWE*. Cryptology ePrint Archive, Report 2014/1018, 2014. http://eprint.iacr.org/2014/1018
10. Albrecht M., Bai S., Ducas L. *A Subfield Lattice Attack on Overstretched NTRU Assumptions*. In: Robshaw M., Katz J. (eds) Advances in Cryptology - CRYPTO 2016. Lecture Notes in Computer Science, vol 9814. Springer, Berlin, Heidelberg, pp 153-178.

11. M. R. Albrecht, R. Player, and S. Scott. *On the concrete hardness of learning with errors*. J. Mathematical Cryptology, 9(3):169-203, 2015. URL: http://www.degruyter.com/view/j/jmc.2015. 9.issue-3/jmc-2015-0016/jmc-2015-0016.xml.
12. Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. *Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator*. In Marc Fischlin and Jean-Sébastien Coron, editors, Advances in Cryptology-EUROCRYPT 2016, volume 9665 of LNCS, pages 789-819. Springer, 2016. https://eprint.iacr.org/2016/146. 20
13. Aharonov, D., Regev, O.: *A lattice problem in quantum NP*. In: FOCS, pp. 210-219 (2003).
14. Ajtai, M.: *The shortest vector problem in L_2 is NP-hard for randomized reductions*. In: STOC, pp. 10-19 (1998).
15. Ambainis, A.: *Quantum walk algorithm for element distinctness*. In: FOCS, pp. 22-31 (2003).
16. S. Arora and Rong Ge. *New algorithms for learning in presence of errors*. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, ICALP 2011, Part I, volume 6755 of LNCS, pages 403-415. Springer, Heidelberg, July 2011.
17. Barbosa, M. et al. (2023). *Fixing and Mechanizing the Security Proof of Fiat-Shamir with Aborts and Dilithium*. In: Handschuh, H., Lysyanskaya, A. (eds) Advances in Cryptology – CRYPTO 2023. CRYPTO 2023. Lecture Notes in Computer Science, vol 14085. Springer, Cham. https://doi.org/10.1007/978-3-031-38554-4_ 12
18. S. Bai, D. Stehlé and W. Wen. *Improved Reduction from the Bounded Distance Decoding Problem to the Unique Shortest Vector Problem in Lattices*. In Springer Proc. of ICALP'2016, pp. 76:1-76:12.
19. S. Bai and S. D. Galbraith. *Lattice decoding attacks on binary LWE* In Willy Susilo and Yi Mu, editors, ACISP 14, volume 8544 of LNCS, pages 322-337. Springer, Heidelberg, July 2014
20. Shi Bai, Austin Beard, Floyd Johnson, Sulani K. B. Vidhanalage, Tran Ngo. *Fiat-Shamir Signatures Based on Module-NTRU*. In Khoa Nguyen, Guomin Yang, Fuchun Guo, Willy Susilo, editors, Information Security and Privacy - 27th Australasian Conference, ACISP 2022, Wollongong, NSW, Australia, November 28-30, 2022, Proceedings. Volume 13494 of Lecture Notes in Computer Science, pages 289-308, Springer, 2022. [doi]
21. A. Becker, L. Ducas, N. Gama, and T. Laarhoven. *New directions in nearest neighbor searching with applications to lattice sieving*. Robert Krauthgamer, editor. Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, DIOP 2016, Arlington, VA, USA, January 10-12, 2016. SIAM, 2016, pages 10-24. https://eprint.iacr.org/2015/1128.
22. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. *Relations among Notions of Security for Public-Key Encryption Schemes*. In Proc. of CRYPTO '98, LNCS 1462, pages 26-45. Springer-Verlag, Berlin, 1998
23. M. Bellare and P. Rogaway. *Random Oracles Are Practical : a Paradigm for Designing Efficient Protocols*. In Proc. of the 1st CCS, pages 62-73. ACM Press, New York, 1993
24. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. *NTRU Prime*. In Jan Camenisch and Carlisle Adams, editors, Selected Areas in Cryptography - SAC 2017, LNCS, to appear. Springer, 2017. http://ntruprime.cr.yp.to/papers.html

25. R. Canetti, O. Goldreich and S. Halevi, *The random oracle methodology*, revisited, STOC'98, ACM, 1998.
26. Devevey, J., Fallahpour, P., Passelègue, A., Stehlé, D. (2023). *A Detailed Analysis of Fiat-Shamir with Aborts*. In: Handschuh, H., Lysyanskaya, A. (eds) Advances in Cryptology – CRYPTO 2023. CRYPTO 2023. Lecture Notes in Computer Science, vol 14085. Springer, Cham. https://doi.org/10.1007/978-3-031-38554-4_ 11
27. Hao Chen, Kristin Lauter, and Katherine E. Stange. *Vulnerable Galois RLWE families and improved attacks*. IACR Cryptology ePrint Archive, 2016. *https://eprint.iacr.org/2016/193*.
28. Yuanmi Chen and Phong Q. Nguyen. *BKZ 2.0: Better lattice security estimates*. In Dong Hoon Lee and Xiaoyun Wang, editors, Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings, volume 7073 of LNCS, pp. 1-20. Springer.T 97, volume 1233 of Lecture Notes in Comput. Sci., pp 52-61. Springer, Berlin, 1997.
29. Chuengsatiansup, Chitchanok et al. "ModFalcon: Compact Signatures Based On Module-NTRU Lattices." Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (2020): n. pag.
30. Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. *Recovering Short Generators of Principal Ideals in Cyclotomic Rings* Marc Fischlin and Jean-Sébastien Coron (Eds.). In Advances in Cryptology Eurocrypt May 2016, Lecture Notes in Computer Science, Springer-Verlag,Proceedings, Part II, pp. pp 559-585.
31. Duman, J., Hövelmanns, K., Kiltz, E., Lyubashevsky, V., Seiler, G., Unruh, D. (2023). *A Thorough Treatment of Highly-Efficient NTRU Instantiations*. In: Boldyreva, A., Kolesnikov, V. (eds) Public-Key Cryptography – PKC 2023. PKC 2023. Lecture Notes in Computer Science, vol 13940. Springer, Cham. https://doi.org/10.1007/978-3-031-31368-4_ 3
32. D. Dadush, O. Regev, and N. Stephens-Davidowitz. *On the closest vector problem with a distance guarantee*. In Proc. of CCC, pages 98-109. IEEE Computer Society Press, 2014.
33. T. El Gamal. *A public key cryptosystem and signature scheme based on discrete logarithms*. IEEE Trans. Inform. Theory, 31:469-472, 1985
34. Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. *A concrete treatment of fiat-shamir signatures in the quantum random-oracle model*. In EUROCRYPT, pages 552–586, 2018.
35. Fouque, P.A., Kirchner, P., Pornin, T., Yu, Y.: *BAT: Small and fast kem over NTRU lattices*. IACR Transactions on Cryptographic Hardware and Embedded Systems 2022(2), 240-265 (Feb 2022)
36. A. Fiat, A. Shamir, *How to prove yourself: practical solutions to identification and signature problems*, Advances in Cryptology—Proceedings of Crypto '86, LNCS, vol. 263, Springer, 1987, pp. 186–194.
37. R. Fujita. *Table of underlying problems of the NIST candidate algorithms.* Available at https://groups.google.com/a/list.nist.gov/d/ msg/pqc-forum/1lDNio0sKq4/7zXvtfdZBQAJ, 2017
38. Nicolas Gama, Malika Izabachène, Phong Q. Nguyen, and Xiang Xie *Structural Lattice Reduction: Generalized Worst-Case to Average-Case Reductions and Homomorphic Cryptosystems*. Marc Fischlin and Jean-Sébastien Coron (Eds.), In Advances in cryptology Eurocrypt May 2016, Lecture Notes in Computer Science, Springer-Verlag Proceedings, Part II, pp. 528-558.

39. Craig Gentry,Chris Peikert,Vinod Vaikuntanathan, STOC '08: Proceedings of the fortieth annual ACM symposium on Theory of computingMay 2008Pages 197–206https://doi.org/10.1145/1374376.1374407
40. Goldwasser S., Micali S. and Rivest R. , A digital signature scheme secure against adaptive chosen- message attacks, SIAM Journal of computing, 17(2), pp. 281-308, April 1988.
41. Grover, L. K.: *A fast quantum mechanical algorithm for database search.* In: STOC, pp. 212-219 (1996)39.
42. Grover, L. K., Rudolph, T.: *How significant are the known collision and element distinctness quantum algorithms?* Quantum Info. Comput.4 (3), pp. 201-206 (2004).
43. Jung Hee Cheon, Jinhyuck Jeong, Changmin Lee *An Algorithm for NTRU Problems and Cryptanalysis of the GGH Multilinear Map without a Low Level Encoding of Zero.* IACR Cryptology ePrint Archive, *https://eprint.iacr.org/2016/139.pdf.*
44. Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. *Terminating BKZ.* IACR Cryptology ePrint Archive report 2011/198, 2011. https://eprint.iacr.org/2011/198.
45. J. Hoffstein, J. Pipher, and J. H. Silverman. *NTRU: A Ring Based Public Key Cryptosystem in Algorithmic Number Theory*, Lecture Notes in Computer Science 1423, Springer-Verlag, pp. 267-288, 1998.
46. Thijs Laarhoven. *Sieving for shortest vectors in lattices using angular locality-sensitive hashing.* In Rosario Gennaro and Matthew Robshaw, editors, Advances in Cryptology CRYPTO 2015 -35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I, volume 9215 of Lecture Notes in Computer Science, pages 3-22. Springer, 2015. https://eprint.iacr.org/2014/744.pdf.
47. Thijs Laarhoven, Michele Mosca, and Joop van de Pol.*Finding shortest lattice vectors faster using quantum search.* Des. Codes Cryptography, 77(2-3):375-400, 2015.
48. M. Liu, X. Wang, G. Xu, and X. Zheng. *A note on BDD problems with $\lambda 2$-gap.* Inf. Process. Lett., 114(1-2):9-12, January 2014.
49. Y. K. Liu, V. Lyubashevsky, and D. Micciancio. *On bounded distance decoding for general lattices.* In Proc. of RANDOM, volume 4110 of LNCS, pages 450-461. Springer, 2006.
50. V. Lyubashevsky, C. Peikert, and O. Regev. *On ideal lattices and learning with errors over rings.* In EUROCRYPT 2010, pages 1-23. Springer, 2010.
51. V. Lyubashevsky, Chris Peikert, and Oded Regev. *A toolkit for ring-LWE cryptography.* In EUROCRYPT 2013, pp. 35-54.
52. V. Lyubashevsky and D. Micciancio. *On bounded distance decoding, unique shortest vectors, and the minimum distance problem.* In Proc. of CRYPTO 2009, pp. 577-594.
53. Vadim Lyubashevsky. *Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures.* In ASIACRYPT, pages 598–616, 2009.
54. Vadim Lyubashevsky. *Lattice signatures without trapdoors.* In EUROCRYPT, pages 738–755, 2012.
55. Qipeng Liu and Mark Zhandry. *Revisiting post-quantum fiat-shamir.* Cryptology ePrint Archive, Report 2019/262, 2019. https://eprint.iacr.org/ 2019/262.
56. Micciancio, D., Voulgaris, P.: *Faster exponential time algorithms for the shortest vector problem.* In DIOP(2010), pp. 1468-1480.

57. D. Moody. *The NIST post quantum cryptography competition.* Available at https://csrc.nist.gov/CSRC/media/Projects/ Post-Quantum-Cryptography/documents/asiacrypt-2017-moody-pqc.pdf, 2017.
58. M. Naor and M. Yung. Public Key Cryptosystems *Provably Secure against Chosen Ciphertext Attacks.* In Proc. of the 22nd ACM STOC, pages 427-437. ACM Press, New York, 1990.
59. NIST Post-Quantum Cryptography- Call for Proposals. Available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization /Call-for-Proposals. List of First Round candidates available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions
60. National Institute of Standards and Technology. *Performance testing of the NIST candidate algorithms.* Available at https://drive.google.com/ file/d/1g-l0bPa-tReBD0Frgnz9aZXpO06PunUa/view, 2017
61. NIST. Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process, jun 2022 https://csrc.nist.gov/Projects/pqc-dig-sig
62. Hiroki OKADA, Atsushi TAKAYASU, Kazuhide FUKUSHIMA, Shinsaku KIYOMOTO and Tsuyoshi TAKAGI *A Compact Digital Signature Scheme Based on the Module-LWR Problem* Journal: IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2021, Volume E104 A, Number 9, Page 1219 DOI: 10.1587/transfun 2020DMP0012
63. Xavier Pujol and Damien Stehlé. *Solving the shortest lattice vector problem in time* $2^{2,465.n}$. IACR Cryptology ePrint Archive, 2009. *https://eprint.iacr.org/2009/605.*
64. C. Peikert. *A useful fact about Ring-LWE that should be known better: it is *at least as hard* to break as NTRU, and likely strictly harder.* Available at http://archive.is/B9KEW.
65. C. Peikert. *Public-key cryptosystems from the worst-case shortest vector problem.* In STOC 2009, pp. 333-342. ACM.
66. C. Peikert and B. Waters. *Lossy trapdoor functions and their applications.* In STOC 2008, pages 187-196, 2008.
67. Pointcheval, D., Stern, J. (1996). Security Proofs for Signature Schemes. In: Maurer, U. (eds) Advances in Cryptology — EUROCRYPT '96. EUROCRYPT 1996. Lecture Notes in Computer Science, vol 1070. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-68339-9_33
68. Regev, O. *On lattices, learning with errors, random linear codes, and cryptography.* In: STOC, pp. 84-93 (2005).
69. O. Regev. *On lattices, learning with errors, random linear codes, and cryptography.* J. ACM, 56(6), 2009.
70. Regev, O.:*Lattices in computer science.* Lecture notes for a course at the Tel Aviv University (2004)78.
71. Regev, O.:*Quantum computation and lattice problems.* SIAM J. Comput. 33 (3), pp. 738-760 (2004).
72. MATZOV:*Report on the Security of LWE: Improved Dual Lattice Attack.* The Center of Encryption and Information Security (2023) https://zenodo.org/record/6412487.
73. Santha, M.: *Quantum walk based search algorithms.* In: TAMC (2008), pp. 31-46 .
74. Schneider, M.: *Analysis of Gauss-Sieve for solving the shortest vector problem in lattices.* In: WALCOM (2011), pp. 89-97.

75. Schneider, M.: *Sieving for short vectors in ideal lattices*. In: AFRICACRYPT (2013), pp. 375-391.
76. C. P. Schnorr and M. Euchner. *Lattice basis reduction: Improved practical algorithms and solving subset sum problems*. Mathematical Programming, 66(1):181-199, 1994
77. Shor, P.W.:*Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*. SIAM J. Comput. 26 (5), pp. 1484-1509 (1997).
78. D. Stehlé and R. Steinfeld. *Making NTRU as secure as worst-case problems over ideal lattices*. Draft of full extended version of Eurocrypt 2011 paper, ver. 10, Oct. 2011. Available at *http://web.science.mq.edu.au*.
79. D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. *Efficient public key encryption based on ideal lattices*. In ASIACRYPT 2009, pp. 617-635. Springer.
80. D. Stehlé and R. Steinfeld. *Making NTRU as secure as worst-case problems over ideal lattices*. In EUROCRYPT 2011, pp. 27-47. Springer.
81. Ludo N. Pulles, Mehdi Tibouch *Cryptanalysis of EagleSign* 2024 https://eprint.iacr.org/2024/1137
82. Wang, X., Liu, M., Tian, C., Bi, J.: *Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem*. In: ASIACCS (2011), pp. 1-9.
83. T. Wunderer. *Revisiting the hybrid attack: Improved analysis and refined security estimates*. Cryptology ePrint Archive, Report 2016/733,2016. http://eprint.iacr.org/2016/733
84. E. Wenger, E. Saxena, M. Malhou, E. Thieu and K. Lauter *Benchmarking Attacks on Learning with Errors* 2024 https://eprint.iacr.org/2024/1229
85. Zhang, J., Feng, D., Yan, D. (2023). *NEV: Faster and Smaller NTRU Encryption Using Vector Decoding*. In: Guo, J., Steinfeld, R. (eds) Advances in Cryptology – ASIACRYPT 2023. ASIACRYPT 2023. Lecture Notes in Computer Science, vol 14444. Springer, Singapore. https://doi.org/10.1007/978-981-99-8739-9_6
86. Zhang, F., Pan, Y., Hu, G.: *A three-level sieve algorithm for the shortest vector problem*. In: SAC (2013), pp. 29-47.

A Implementation of EagleSignV3

A.1 Constant time implementation for EagleSignV3

As Dilithium, we do not use branch depending on secret data and also we do not use access memory locations that depend on secret data. Moreover, for modular reductions mod q, we do not use the '%' operator of the C programming language, instead we use Montgomery reductions. We do not use also rejection sampling in the signature and verification algorithm.

A.2 Bit/Byte Packing for EagleSignV3

In this section, we will explain the process of converting vectors and matrices into byte strings and vis-versa. The procedure used in our implementation is similar to the one used in Dilithium round 3. For completeness purpose, we will describe it in this section. The general rule that we will follow is that if the range of an element x consists exclusively of non-negative integers, then we simply pack the integer x. If x is from a

range $[-a, b]$ that may contain some negative integers, then we pack the positive integer $b - x$.

Let's start with a single polynomial of N coefficients $N \in \{256, 512, 1024, 2048\}$ where each coefficient is an integer which can be encoded on b bits. Then each set of 8 coefficients can be encoded on $8 * b/8 = b$ bytes.

In the case of EagleSignV3 signature $(r, \mathbf{z})$ or $(r, \mathbf{z}, \mathbf{f})$, r is a byte array and don't need any conversion. $\mathbf{z}$ in EagleSignV3 is a vector of l elements $l \in \{1, 2, 3, 4\}$ where each polynomial's coefficient can be encoded on $\log_2(2 \times \gamma_1)$ [resp: $\log_2(2 \times \delta_C + 1)$, $\log_2(2 \times \delta_D + 1)$] bits. This means that each set of 8 coefficients of $\mathbf{z}$ polynomials can be encoded on $\log_2(2 \times \gamma_1)$ bytes string.

The previous described procedure has also been used to pack and unpack different other parameters including the matrix $\mathbf{T}$ [resp: vector $\mathbf{t}$] in the public key of EagleSignV3 as well as $\mathbf{D}, \mathbf{G}$ in the corresponding private key. In appendix B, we have provided a python code that we wrote in order to generate the set of instructions in C language to convert a list of 8 different b-bits coefficients into a bytes string for any integer $b > 1$.

A.3 NTT transformation for EagleSignV3

It is known that we can perform multiplication and division efficiently over $R_q = \dfrac{\mathbb{Z}_q(X)}{m(X)}$ whenever the polynomial $m(x)$ factors into a product of s polynomials $m(X) = \pi_{i=1}^{s} m_i(X)$ having small degree assuming that $m_i(X)$ is coprime with $m_j(X)$ for all $i, j, \neq j$. In this case, by Chinese Remainder Theorem we have an isomorphism $\phi : \dfrac{\mathbb{Z}_q(X)}{m(X)} \cong \dfrac{\mathbb{Z}_q(X)}{m_1(X)} \times \ldots \times \dfrac{\mathbb{Z}_q(X)}{m_s(X)}$ wit an efficient algorithm for both ϕ and ϕ^{-1} and thus multiplication and division in $R_q = \dfrac{\mathbb{Z}_q(X)}{m(X)}$ can be transferred in $\dfrac{\mathbb{Z}_q(X)}{m_i(X)}$.

For EagleSignV3, with $q = 59393,249857$ and $n = 512, 1024, 2048$, we use the classical common friendly NTT ring is $R_q = \dfrac{\mathbb{Z}_q(X)}{m(X)}$ where $m(X) = X^n + 1$, $n = 2^u$, q is prime and $q = 1 \mod 2n$.

A.4 Hashing and Sampling techniques, special functions for EagleSignV3

Sampling y : The function VectorUnifEtaPoly$(\lambda, \mathcal{K})$ maps $(\lambda, \mathcal{K})$ to $\mathbf{y} \in S_{\gamma_1}^{l}$. We compute independently the l components of y. Note that these components are polynomials in S_{γ_1}. For the i-th polynomial, $0 \leq i < l$, it absorbs the 48 bytes of λ concatenated with the 2 bytes representing $\mathcal{K} + i$ in little endian byte order into SHAKE-256.

Sampling invertible G : For EagleSignV3, the function MatrixUnifEtaPolyn(β_1, η, l, l) maps (β_1, η, l, l) to $\mathbf{G} \in S_{\eta_G}^{l \times l}$. We compute independently the $l \times l$ components of $\mathbf{G}$. For each polynomial $\mathbf{G}_{(i,j)}$, $0 \leq i, j < l$, it absorbs the 96 bytes of β_1 concatenated with the 2 bytes representing $i \times l + j$ in little endian byte order into SHAKE-256. If $\mathbf{G}$ is not invertible in $\mathbf{R}_q^{l \times l}$, we renew the seed β_1 by computing $\beta_1 =$ SHAKE-256(β_1) until $\mathbf{G}$ is invertible. Remark that this algorithm terminates quickly since the ring R_q contains enough invertible polynomials

Sampling D : For EagleSignV3 , the function MatrixUnifEtaPolyn(β_2, η_D, k, l) maps (β_2, η_D, k, l) to $\mathbf{D} \in S_{\eta_D}^{k \times l}$. We compute independently the $k \times l$ components of $\mathbf{D}$. For each polynomial $\mathbf{D}_{(i,j)}$, $0 \leq i < l, 0 \leq j < k$, it absorbs the 96 bytes of β_2 concatenated with the 2 bytes representing $i \times l + j$ in little endian byte order into SHAKE-256.

Computing c:

The cryptographic Hash function H maps (μ, r) to $\mathbf{c} \in B_\tau^l$. For this purpose we first extract 640 bits of the output of SHAKE-256 onto the input μ, r in this order as a seed seed_c. We then compute independently the l components of $\mathbf{c}$. For each polynomial $\mathbf{c}_i$, $0 \leq i < l$, we absorb the 80 bytes of seed_c concatenated with the 2 bytes representing i in little endian byte order into XOF interpreted as SHAKE/STREAM-128 of the FIPS202 standard. The output of the XOF is used to generate $\mathbf{c}_i = d$ in a Ball as follows:

- Initialize $d = d_0 d_1 \dots d_{N-1} = 0 \dots 0$
- for $i = N - \tau$ to N
  - $b \xleftarrow{\$} \{0, 1, \dots, i\}$ with XOF
  - $d_i := d_b$
  - $s \xleftarrow{\$} \{0, 1\}$ with XOF
  - $d_b := 1 - 2s$
- return d

Note that in the expression $\mathbf{c}_i = d$, d is used to simplify the notation in the previous algorithm.

Sampling the Matrix A : For EagleSignV3, the function MatrixUnifEtaPolyn maps a uniform seed $\rho \in \{0,1\}^{256}$ to a matrix $\mathbf{A} \in R_q^{k \times l}, q, N$ such that $q \cong 1 \mod 2n$ in NTT domain representation. $\mathbf{A}$ is generated and stored in NTT representation as $\hat{\mathbf{A}}$. We computes independently the components $\hat{\mathbf{a}}_{i,j} \in R_q$ of $\hat{\mathbf{A}}$. We use SHAKE-128 to compute the coefficient $\hat{\mathbf{a}}_{i,j}$ by absorbing the 32 bytes of ρ followed by 2 bytes representing $0 \leq 2^8 \times i + j < 2^{16}$ in little-endian byte order. The output stream of SHAKE-128 is interpreted as a sequence of integers between 0 and $2^{|q|} - 1$, where $|q|$ is the bit-size of prime q which is used. To obtain such result for EagleSignV3, we do the following.

- For $q = 59393, |q| = 16$, we interpreting blocks of 2 consecutive bytes in little endian byte order. In practice, the two consecutive bytes b_0 , b_1 are used to get the integer $0 \leq t' = b_1 \times 2^8 + b_0 \leq 2^{16} - 1$ and compute t as the logical AND of $t' = b_1 \times 2^8 + b_0$ and $2^{16} - 1$.
- For $q = 249857, |q| = 18$, we set the six highest bits of every third byte to zero and interpreting blocks of 3 consecutive bytes in little endian byte order. In practice, the three consecutive bytes b_0 , b_1 , b_2, are used to get the integer $0 \leq t = b_2' \times 2^{16} + b_1 \times 2^8 + b_0 \leq 2^{18} - 1$ where b_2' is the logical AND of b_2 and $2^6 - 1$. Another method is to compute t as the logical AND of $t' = b_2 \times 2^{16} + b_1 \times 2^8 + b_0$ and $2^{18} - 1$.

Finally, MatrixUnifEtaPolyn performs rejection sampling on these $|q|$-bit integers t to sample the N coefficients between 0 and $q - 1$.

Collision resistant hash (CRH1, CRH) For EagleSignV3, the function CRH1 and CRH are collision resistant hash functions. For this purpose 256 and 384 bits of the output of SHAKE-256 are used respectively for CRH1 and CRH. Note that we can easily choose and integrate other hash functions.
CRH1 is called on the public key $(\rho, \mathbf{T})$ and $(\rho, \mathbf{t})$to compute tr. For this reason, it takes as input the byte string obtained from packing ρ and $\mathbf{T}$ (resp: $\mathbf{t}$) in this order and the result is absorbed into SHAKE-256 and the first 32 output bytes are used as the resulting hash.

CRH on the other hand is called on the input $tr||M$ to compute μ. Here the concatenation of the hash tr and the message string M are absorbed into SHAKE-256 and the first 48 output bytes are used as the resulting hash.

Collision resistant hash For EagleSignV3, the functions G, G_1, G_2 are collision resistant hash functions. For this purpose 256 bits of the output of SHAKE-256 is used. G, G_1 or G_2 is called the input $\mathbf{p}$ to compute r in v to compute r' in the verification algorithm. Note that, we can easily choose and integrate other hash functions.

B Conversion

In this appendix B, we have provided a python code that we wrote in order to generate the set of instructions in C language to convert a list of 8 different b-bits coefficients into a bytes string for any integer $b > 1$.

Bit-Packing for EagleSignV3.
Python Code for generating Bit-Packing instructions in C

```
import numpy as np
import pandas as pd

def pack(D, Dp, dtype="int32_t", dterm="logeta", n=8):
  if Dp%2 == 0 and n>1:
    return pack(D, Dp//2, dtype, dterm, n//2)


  X = [[i]*D for i in range(8)]
  Y = [[-1]*8 for i in range(D)]
  Z= [-1]*(8*D)

  l = 0
  for i in range(8):
    for j in range(D):
      Z[l] = X[i][j]
      l += 1

  l = 0
  for i in range(D):
    for j in range(8):
```

```
      Y[i][j] = Z[l]
      l += 1

  ta = []
  tb = []
  for y in Y:
    y = pd.Series(y)
    c = dict(y.value_counts())
    ta.append(c)
    for key in c.keys():
      tb.append({key: c[key]})

  print("\nunsigned int i;\n{} t[{}]; \nfor(i=0;i<N/{};++i)\n
    {{\n".format(dtype, n, n))

      for i in range(n):
        print("    t[{0}] = (1 << ({1} - 1)) - a->coeffs[{2} *
     i + {0}];".format(i, dterm, n))
      print()

      cp = 0
      cp_key = 0
      it = 0
      for y in Y:
        y = pd.Series(y)
        c = dict(y.value_counts())
        init = 0
        sorted_ = list(c.keys())
        sorted_.sort()
        for key in sorted_:
          if key >= n:
          break
          cp = cp%D
          if init == 0:
          print("    r[{} * i + {}] = t[{}]{};".format(Dp, it,
     key, " >> {}".format(cp) if cp else ""))
          init += c[key]
          else:
          print("    r[{} * i + {}] {}= t[{}]{};".format(Dp,
    it, "|" if init else "", key, " << {}".format(init) if
    init else ""))
          init += c[key]


          if (cp_key == key):
          cp += c[key]
          else:
          cp = c[key]

          cp_key = key
```

```
67
68          it += 1
69        print("\n}")
70
71  if __name__ == "__main__":
72    D = 21  # Change this value according to your need
73    dterm = "COEFF_BIT_SIZE"
74    pack(D, D, dterm)
```

The out of the previous code is presented in the next code. Note that the output generated depends on four (04) parameters : N, r, a and $COEFF\_BIT\_SIZE$. r is the output byte array, a is the input polynomial, N is the number of components in polynomial a, $N \in \{256, 512, 1024, 2048\}$ and $COEFF\_BIT\_SIZE$ is the coefficients' bits size.

```
1  unsigned int i;
2  Q_SIZE t[8];
3  for(i=0;i<N/8;++i)
4  {
5
6      t[0] = (1 << (COEFF_BIT_SIZE - 1)) - a->coeffs[8 * i + 0];
7      t[1] = (1 << (COEFF_BIT_SIZE - 1)) - a->coeffs[8 * i + 1];
8      t[2] = (1 << (COEFF_BIT_SIZE - 1)) - a->coeffs[8 * i + 2];
9      t[3] = (1 << (COEFF_BIT_SIZE - 1)) - a->coeffs[8 * i + 3];
10     t[4] = (1 << (COEFF_BIT_SIZE - 1)) - a->coeffs[8 * i + 4];
11     t[5] = (1 << (COEFF_BIT_SIZE - 1)) - a->coeffs[8 * i + 5];
12     t[6] = (1 << (COEFF_BIT_SIZE - 1)) - a->coeffs[8 * i + 6];
13     t[7] = (1 << (COEFF_BIT_SIZE - 1)) - a->coeffs[8 * i + 7];
14
15     r[21 * i + 0] = t[0];
16     r[21 * i + 1] = t[0] >> 8;
17     r[21 * i + 2] = t[0] >> 16;
18     r[21 * i + 2] |= t[1] << 5;
19     r[21 * i + 3] = t[1] >> 3;
20     r[21 * i + 4] = t[1] >> 11;
21     r[21 * i + 5] = t[1] >> 19;
22     r[21 * i + 5] |= t[2] << 2;
23     r[21 * i + 6] = t[2] >> 6;
24     r[21 * i + 7] = t[2] >> 14;
25     r[21 * i + 7] |= t[3] << 7;
26     r[21 * i + 8] = t[3] >> 1;
27     r[21 * i + 9] = t[3] >> 9;
28     r[21 * i + 10] = t[3] >> 17;
29     r[21 * i + 10] |= t[4] << 4;
30     r[21 * i + 11] = t[4] >> 4;
31     r[21 * i + 12] = t[4] >> 12;
32     r[21 * i + 13] = t[4] >> 20;
33     r[21 * i + 13] |= t[5] << 1;
34     r[21 * i + 14] = t[5] >> 7;
35     r[21 * i + 15] = t[5] >> 15;
36     r[21 * i + 15] |= t[6] << 6;
```

```
    r[21 * i + 16] = t[6] >> 2;
    r[21 * i + 17] = t[6] >> 10;
    r[21 * i + 18] = t[6] >> 18;
    r[21 * i + 18] |= t[7] << 3;
    r[21 * i + 19] = t[7] >> 5;
    r[21 * i + 20] = t[7] >> 13;

}
```

Bit-Unpacking for EagleSignV3.
Python Code for generating Bit-Unpacking instructions in C

```
import numpy as np
import pandas as pd

def unpack(D, Dp, Ty="int32_t", dterm="logeta", n=8):
    if Dp%2 == 0 and n>1:
        return unpack(D, Dp//2, Ty, dterm, n//2)

    X = [[i]*D for i in range(8)]
    Y = [[-1]*8 for i in range(D)]
    Z= [-1]*(8*D)
    l = 0
    for i in range(8):
        for j in range(D):
            Z[l] = X[i][j]
            l += 1

    l = 0
    for i in range(D):
        for j in range(8):
            Y[i][j] = Z[l]
            l += 1

    ta = []
    tb = []
    for y in Y:
        y = pd.Series(y)
        c = dict(y.value_counts())
        ta.append(c)
        for key in c.keys():
            tb.append({key: c[key]})

    print("\nunsigned int i;\nfor(i=0;i<N/{};++i)\n{{\n".
    format(n))

    cp = 0
    cp_key = 0
    it = 0
    for y in Y:
        y = pd.Series(y)
```

```
        c = dict(y.value_counts())
        init = 0
        sorted_ = list(c.keys())
        sorted_.sort()
        for key in sorted_:
            if key >= n:
                break
            cp = cp%D
            if init == 0:
                print("    r->coeffs[{} * i + {}] {}= {}a[{} * i + {}]{};".format(n, key, "|" if cp else "", "(%s)"%(Ty) if cp else "", Dp, it, " << {}".format(cp) if cp else "")
)
                init += c[key]
            else:
                print("    r->coeffs[{} * i + {}] &= {};\n".format(n, cp_key, hex((2<<(D-1))-1)))
                print("    r->coeffs[{} * i + {}] = a[{} * i + {}]{};".format(n, key, Dp, it, " >> {}".format(init) if init else ""))
                init += c[key]


            if (cp_key == key):
                cp += c[key]
            else:
                cp = c[key]

            cp_key = key

        it += 1
    print("    r->coeffs[{} * i + {}] &= {};\n".format(n, cp_key, hex((2<<(D-1))-1)))


    for i in range(n):
        print("    r->coeffs[{2} * i + {0}] = (1 << ({1} - 1)) - r->coeffs[{2} * i + {0}];".format(i, dterm, n))

    print("\n}")

if __name__=="__main__":
  D = 21  # Change this value according to your need
  dataType = "int32_t"
  dterm = "COEFF_BIT_SIZE"
  unpack(D, D, dataType, dterm)
```

The out of the previous code is presented in the next code. Note that the output generated depends on three (04) parameters : N, r, a and $COEFF\_BIT\_SIZE$. a is the input byte array, r is the output polynomial, N is the number of components in

polynomial r, $N \in \{256, 512, 1024, 2048\}$ and $COEFF\_BIT\_SIZE$ is the coefficients' bits size.

```
unsigned int i;
for(i=0;i<N/8;++i)
{

    r->coeffs[8 * i + 0] = a[21 * i + 0];
    r->coeffs[8 * i + 0] |= (int32_t)a[21 * i + 1] << 8;
    r->coeffs[8 * i + 0] |= (int32_t)a[21 * i + 2] << 16;
    r->coeffs[8 * i + 0] &= 0x1fffff;

    r->coeffs[8 * i + 1] = a[21 * i + 2] >> 5;
    r->coeffs[8 * i + 1] |= (int32_t)a[21 * i + 3] << 3;
    r->coeffs[8 * i + 1] |= (int32_t)a[21 * i + 4] << 11;
    r->coeffs[8 * i + 1] |= (int32_t)a[21 * i + 5] << 19;
    r->coeffs[8 * i + 1] &= 0x1fffff;

    r->coeffs[8 * i + 2] = a[21 * i + 5] >> 2;
    r->coeffs[8 * i + 2] |= (int32_t)a[21 * i + 6] << 6;
    r->coeffs[8 * i + 2] |= (int32_t)a[21 * i + 7] << 14;
    r->coeffs[8 * i + 2] &= 0x1fffff;

    r->coeffs[8 * i + 3] = a[21 * i + 7] >> 7;
    r->coeffs[8 * i + 3] |= (int32_t)a[21 * i + 8] << 1;
    r->coeffs[8 * i + 3] |= (int32_t)a[21 * i + 9] << 9;
    r->coeffs[8 * i + 3] |= (int32_t)a[21 * i + 10] << 17;
    r->coeffs[8 * i + 3] &= 0x1fffff;

    r->coeffs[8 * i + 4] = a[21 * i + 10] >> 4;
    r->coeffs[8 * i + 4] |= (int32_t)a[21 * i + 11] << 4;
    r->coeffs[8 * i + 4] |= (int32_t)a[21 * i + 12] << 12;
    r->coeffs[8 * i + 4] |= (int32_t)a[21 * i + 13] << 20;
    r->coeffs[8 * i + 4] &= 0x1fffff;

    r->coeffs[8 * i + 5] = a[21 * i + 13] >> 1;
    r->coeffs[8 * i + 5] |= (int32_t)a[21 * i + 14] << 7;
    r->coeffs[8 * i + 5] |= (int32_t)a[21 * i + 15] << 15;
    r->coeffs[8 * i + 5] &= 0x1fffff;

    r->coeffs[8 * i + 6] = a[21 * i + 15] >> 6;
    r->coeffs[8 * i + 6] |= (int32_t)a[21 * i + 16] << 2;
    r->coeffs[8 * i + 6] |= (int32_t)a[21 * i + 17] << 10;
    r->coeffs[8 * i + 6] |= (int32_t)a[21 * i + 18] << 18;
    r->coeffs[8 * i + 6] &= 0x1fffff;

    r->coeffs[8 * i + 7] = a[21 * i + 18] >> 3;
    r->coeffs[8 * i + 7] |= (int32_t)a[21 * i + 19] << 5;
    r->coeffs[8 * i + 7] |= (int32_t)a[21 * i + 20] << 13;
    r->coeffs[8 * i + 7] &= 0x1fffff;
```

```
48
49      r->coeffs[8 * i + 0] = (1 << (COEFF_BIT_SIZE - 1)) - r->
        coeffs[8 * i + 0];
50      r->coeffs[8 * i + 1] = (1 << (COEFF_BIT_SIZE - 1)) - r->
        coeffs[8 * i + 1];
51      r->coeffs[8 * i + 2] = (1 << (COEFF_BIT_SIZE - 1)) - r->
        coeffs[8 * i + 2];
52      r->coeffs[8 * i + 3] = (1 << (COEFF_BIT_SIZE - 1)) - r->
        coeffs[8 * i + 3];
53      r->coeffs[8 * i + 4] = (1 << (COEFF_BIT_SIZE - 1)) - r->
        coeffs[8 * i + 4];
54      r->coeffs[8 * i + 5] = (1 << (COEFF_BIT_SIZE - 1)) - r->
        coeffs[8 * i + 5];
55      r->coeffs[8 * i + 6] = (1 << (COEFF_BIT_SIZE - 1)) - r->
        coeffs[8 * i + 6];
56      r->coeffs[8 * i + 7] = (1 << (COEFF_BIT_SIZE - 1)) - r->
        coeffs[8 * i + 7];
57
58  }
```