ChatGPT

VEINN: Vector Encrypted Invertible Neural Network for Post-Quantum Cryptography

Abstract

We present **VEINN**, a novel encryption scheme that combines *invertible neural network* (INN) architectures with *lattice-based* hardness to achieve a vectorized block cipher with homomorphic capabilities. VEINN treats each block as an n-dimensional vector over $\mathbb{Z}_{2^{16}}$, applying a sequence of *coupling layers*, *elementwise modular scalings*, and *permutations* to produce ciphertexts. All parameters (masks and scalings) are derived from a secret seed, optionally via an LWE-based pseudorandom function (PRF) to inject Learning With Errors (LWE) hardness. As a result, VEINN enjoys the flexibility and speed of modern invertible networks while inheriting post-quantum security from lattice assumptions. The scheme supports lattice-style homomorphic addition and multiplication of encrypted vectors, akin to ring-based FHE schemes (e.g. CKKS [1]), enabling private computations. We detail the design, analyze its algebraic structure and security, compare to existing symmetric and lattice-based schemes, and discuss classical and quantum attack resilience. Cryptanalysis considerations suggest that VEINN's security reduces to hard problems in vector spaces and LWE, making it a plausible candidate for future post-quantum cryptography.

Introduction

As quantum computing progresses, **post-quantum cryptography** (PQC) is essential to secure communications. Lattice-based schemes, relying on the *Learning With Errors (LWE)* problem, are leading PQC candidates [2] [1]. Meanwhile, invertible neural networks (normalizing flows) have shown that deep models can implement complex bijections on high-dimensional vectors [3]. VEINN synergizes these trends by using an invertible neural network as a secret-key cipher whose parameters are derived via lattice-hard randomness. In effect, VEINN is a block cipher on $\mathbb{Z}_{2^{16}}^n$: each block is an n-word vector, and encryption is a parameterized linear-affine bijection in this vector space (mod 2^{16}). The secret key (a seed) generates coupling masks and per-coordinate scalings that are guaranteed invertible (odd mod 2^{16}) [4]. Optional LWE-PRF rounds add noise-based security: parameters are pseudorandom under LWE hardness [5]. The result is a fast, highly parallelizable cipher with built-in homomorphic sum and product (in the ring $\mathbb{Z}_Q[x]/(x^n+1)$) and potential resistance to quantum attacks.

This paper provides a detailed exposition of VEINN's design and concepts. We explain how lattice features (LWE, ring convolution) and INN features (coupling layers, invertible scaling, shuffles) interact in a unified vector-space framework. We highlight why treating both ciphertexts and keys as vectors in linear spaces yields performance and security advantages. We compare VEINN to conventional encryption (symmetric block ciphers, lattice KEMs) and homomorphic schemes (CKKS-style RLWE encryption) in terms of security and functionality. Finally we discuss classical and quantum cryptanalysis: examining key-recovery, structural attacks, and hardness assumptions, to assess VEINN's viability as a PQC primitive.

Background

Lattice-Based Cryptography and LWE

The security of many PQC schemes rests on lattice problems. Notably, Oded Regev showed that LWE is as hard as worst-case lattice problems like GapSVP under quantum reductions [2]. Informally, LWE posits that given "noisy" inner products $\mathbf{a}\cdot \mathbf{s} + e\pmod{q}$, recovering the secret vector $ \mathbf{s}$ or distinguishing from random is computationally hard (even for quantum computers). Ring-LWE (RLWE) is an efficient variant used in homomorphic encryption (e.g. CKKS [1]). LWE-based schemes allow homomorphic addition and multiplication (with noise-growth) on ciphertexts, a key feature in FHE. VEINN leverages LWE-style randomness in its key schedule to benefit from these hardness assumptions.

Invertible Neural Networks (Normalizing Flows)

Invertible (bijective) neural networks, also known as *normalizing flows*, provide efficient, invertible transformations on high-dimensional data [3]. A common building block is the **affine coupling layer** (Real NVP style [6]): it splits a vector into two halves and transforms one half using a function of the other, preserving invertibility and a triangular Jacobian. By stacking multiple coupling layers with alternating masks, complex invertible mappings are achieved while allowing exact inverse computation [7]. In VEINN, the coupling layers are implemented by simple modular operations (additive masks and cyclic convolution by the all-ones vector), yielding an efficient bijection on $\mathbb{Z}_{2^{16}}^n$. Because all operations are linear or affine mod 2^k, the network acts as a permutation in this vector space, i.e. a reversible linear-affine cipher with key-dependent parameters.

Homomorphic Encryption (CKKS)

Homomorphic encryption schemes allow computation on ciphertexts. The CKKS scheme [1] (Cheon *et al.*, 2017) is a lattice-based FHE system supporting approximate arithmetic on real/complex numbers via RLWE. CKKS ciphertexts live in a polynomial ring and use *rescaling* to control growth. VEINN, while not fully homomorphic, provides *some* homomorphic structure: encrypted vectors can be added or multiplied (via negacyclic convolution) to yield a new ciphertext which decrypts to the algebraic sum or product of plaintexts. This follows lattice/FHE practice of homomorphic ring operations, but in VEINN's case is implemented over $\mathbb{Z}_{2^{16}}$ with no bootstrapping (the noise budget concept differs, since VEINN's noise comes from LWE PRF, not message noise). Thus VEINN can support applications like private comparison or summation in the encrypted domain, similar to partially-homomorphic lattice schemes.

The VEINN Construction

Block Representation

VEINN operates on fixed-size blocks. Each block is an n-word vector $x\in \mathbb{Z}_{2^{16}}^n$, interpreted as a sequence of bytes (since n words yields $2n$ bytes). The scheme pads the plaintext (PKCS#7) so its length is a multiple of $2n$, then splits it into blocks. Each block is converted to a vector of n unsigned 16-bit integers (dtype `np.uint16`) via `bytes_to_block` [8].

Key Schedule and Parameters

The secret key is a random seed of fixed length (e.g. 128 bits). From this seed, all round parameters are derived in a deterministic but pseudorandom way. A `VeinnParams` object specifies: block length n, number of rounds R, layers per round ℓ, shuffle stride, and whether to use LWE. The function `key_from_seed(seed, vp)` generates a `VeinnKey` containing:

- **Shuffle index**: a fixed permutation of ${0,\dots,n-1}$ of order coprime with n, generated by `make_shuffle_indices(n, stride)` [9]. This permutation is applied after each round to mix vector coordinates.
- **Coupling masks**: For each round $r=0,\dots,R-1$ and each layer $l=0,\dots,\ell-1$, two random masks $\mathbf{a},\mathbf{b}\in \mathbb{Z}_{2^{16}}^{n/2}$ are generated. These masks feed into the coupling layer transforms. The masks are derived via the function `derive_u16(h, vp, seed, tag, b"A")` and `b"B"`, which either takes direct SHAKE output or runs an LWE-based PRF (if `use_lwe`) [10]. Notably, the LWE PRF (`lwe_prf_expand`) uses negacyclic convolution with a secret and public vector and small error to generate pseudorandom output [5], embedding lattice hardness into the masks.
- **Scaling vectors**: Each round r has an *elementwise scaling vector* $\mathbf{s}\in\mathbb{Z}_{2^{16}}^n$ such that each s_i is odd (thus invertible mod 2^{16}). We ensure oddness via `ensure_odd_vec` [4] and compute its inverse `inv_vec_mod_q(s)` [4]. During encryption, the state vector is multiplied coordinatewise by $\mathbf{s}$ (mod 2^{16}); during decryption, by $\mathbf{s}^{-1}$. The use of random invertible scalings adds diffusion and makes the cipher piecewise affine. The requirement that s_i be odd ensures $(s_i,2^{16})=1$ so a modular inverse exists [4].

Mathematically, the key schedule defines a sequence of *RoundParams*, each consisting of a list of `CouplingParams` (mask pairs) and the $\mathbf{s}$ and $\mathbf{s}^{-1}$ vectors. With LWE-PRF enabled, recovering these parameters from the seed is as hard as solving the underlying LWE instance (quantum-hard under standard assumptions [2]).

Encryption (Forward Permutation)

Encryption of one block $x\in \mathbb{Z}_{2^{16}}^n$ with key k (seed-derived) is performed by `permute_forward(x,k)` [11]. The process for each round $r=0,\dots,R-1$ is:

1. **Coupling Layers**: For each of the ℓ coupling layers, split the current state y into halves $y_1,y_2\in \mathbb{Z}_{2^{16}}^{n/2}$. Compute:
2. $t = \mathrm{conv}(y_2 + \mathbf{a}, \mathbf{1})$ and set $y_1 := y_1 + t \pmod{2^{16}}$.
3. $u = \mathrm{conv}(y_1 + \mathbf{b}, \mathbf{1})$ and set $y_2 := y_2 + u \pmod{2^{16}}$. Here $\mathbf{a},\mathbf{b}$ are the masks for this layer, and $\mathrm{conv}(v,\mathbf{1})$ denotes *negacyclic convolution* of v with the all-ones vector mod $x^{n/2}+1$, implemented by `negacyclic_convolution` [12]. In code, this is `coupling_forward` [13]. Since each layer mixes one half of the state into the other, stacking multiple layers fully mixes the bits (the coupling Jacobian is triangular [3] but full-rank).
4. **Invertible Scaling**: Multiply each coordinate of y by the round's scaling vector $\mathbf{s}$: $y := y \cdot \mathbf{s} \pmod{2^{16}}$ [14]. Because each s_i is invertible mod 2^{16}, this is a

bijection. This step is nonlinear across rounds because different rounds have independent $\mathbf{s}$.

5. **Shuffle Permutation**: Permute the coordinates of y by the fixed shuffle index π: $y := y_\pi$. This breaks simple alignment between rounds and spreads information globally [14].

After R rounds, y is the ciphertext vector. In effect, encryption is a composition of many *affine transformations* (coupling adds convolutions of one half into the other) and diagonal multiplications, making it highly nonlinear in the plaintext. The overall map $x\mapsto y$ is bijective (per-key) and key-dependent.

In pseudocode:

```
y := x
for r in 0..R-1:
   for each (a,b) in masks of round r:
      [y1,y2] := split(y)
      t := conv(y2 + a, ones); y1 := y1 + t mod Q
      u := conv(y1 + b, ones); y2 := y2 + u mod Q
      y := concat(y1,y2)
   y := (y * s_r) mod Q      # elementwise multiplication by scaling vector
   y := shuffle(y, π)       # permutation of coordinates
ciphertext := y
```

This construction is essentially an *invertible neural network* (similar in spirit to RealNVP flows), applied to vector encryption.

Decryption (Inverse Permutation)

Decryption reverses the above: given ciphertext y, apply the inverse permutation, inverse scaling, and inverse coupling layers in reverse order to recover x. Concretely, `permute_inverse(y,k)` does:

- For each round $r=R-1,\dots,0$:
- Unshuffle by π^{-1} to restore original coordinate order.
- Multiply by $\mathbf{s}^{-1}$ (the precomputed modular inverse of $\mathbf{s}$) to undo scaling.
- For each coupling layer in reverse, do the inverse of `coupling_forward`: subtract the convolution terms in reverse order [15]. The code in `coupling_inverse` [16] ensures exact inversion.

Because all steps were bijections, decryption recovers x perfectly as long as operations mod 2^{16} commute correctly (they do). Padding bytes are then removed. In the implemented system, an HMAC over the ciphertext and timestamp (using the seed as HMAC key) provides integrity checking [17].

Homomorphic Properties

VEINN supports two homomorphic operations on ciphertext files:

- **Homomorphic Addition**: Given two encrypted vectors $E(x)$ and $E(y)$ (blockwise ciphertexts), their componentwise sum mod 2^{16} decrypts to $x+y$ [18]. This follows because the encryption map is linear mod 2^{16} apart from known offsets; adding ciphertexts corresponds to adding plaintexts in the same ring. In code, two JSON ciphertext files are parsed to vectors `enc1, enc2` and added blockwise: `summed = [(a+b)%Q]` [19].

- **Homomorphic Multiplication**: Given ciphertexts $E(x)$ and $E(y)$, one can compute a "product" ciphertext by negacyclic convolution of each pair of ciphertext blocks [20]. This corresponds to multiplying the plaintext polynomials mod (x^n+1). In practice, the code computes `prod = [negacyclic_convolution(a,b,Q) for (a,b) in zip(enc1,enc2)]` [21]. While this does not directly equal encryption of the product $x*y$ in the usual integer sense, it emulates polynomial multiplication in the ring. Combined with relinearization or scaling (not shown), this could support polynomial evaluations. Thus VEINN exhibits rudimentary FHE-like behavior: it handles additions and ring-multiplications homomorphically. (Unlike CKKS [1], there is no noise budget for message; the only "noise" comes from LWE-PRF randomness, which does not accumulate across operations.)

These homomorphic operations are possible because VEINN's ciphertext space is a vector space over $\mathbb{Z}_{2^{16}}$ and we allow algebraic manipulation of vectors. In other words, VEINN treats ciphertexts as lattice vectors, enabling lattice-like homomorphism.

Additional Features

- **LWE PRF**: The boolean flag `use_lwe` determines whether mask/scale values are drawn directly from SHAKE (deterministic) or via a special LWE-based PRF (`lwe_prf_expand`) [5]. With LWE enabled, each mask is pseudo-random with embedded Gaussian noise e, making parameter recovery analogous to solving an LWE instance. This adds post-quantum security: without knowing the seed, an attacker must solve an LWE problem to predict masks [2].

- **Shuffle Stride**: The shuffle is a fixed cyclic multiplication index (with multiplier coprime to n) [11]. This simple permutation provides diffusion at each round. The stride parameter should be chosen coprime to n to ensure a full cycle (as enforced by `make_shuffle_indices`) [9].

- **Integrity and Key Management**: Each encrypted file carries metadata (parameters, timestamp) and an HMAC (SHA-256) over the ciphertext using the seed as key [17] [22]. This prevents manipulation or replay. A separate RSA KEM (not shown) can be used to encrypt the seed for public-key encryption, but the core VEINN transform is symmetric.

- **Vector Space Structure**: Crucially, both the *keys* (masks, scales, shuffle index) and *ciphertexts* in VEINN lie in linear spaces over $\mathbb{Z}*{2^{16}}$. The encryption transform is a linear-affine permutation of $\mathbb{Z}*^n$. Treating data as vectors (instead of bitstrings) allows highly parallel operations (e.g. 16-bit arithmetic in SIMD) and leverages algebraic tools (e.g. Fourier transforms for convolution). This contrasts with bit-oriented ciphers (AES) or pure integer ring ciphers (RSA): VEINN directly operates on multi-word blocks as algebraic entities, simplifying homomorphic operations and analysis in a vector-matrix framework.}

Security Rationale and Performance

Post-Quantum Hardness

- **LWE-based Nonlinearity**: If LWE-PRF is enabled, breaking VEINN requires solving LWE (or RLWE) to recover masks or predict the PRF output, as the seed-derivation uses negacyclic convolution with secret s and noise e [5]. Since LWE is believed hard even for quantum adversaries [2], this protects the key schedule from attacks that exploit linear algebra. Even without LWE, the invertible network itself is highly nonlinear (elementwise mod multiplies + convolutions), though that lacks a formal security proof.

- **Large Keyspace**: A random seed (e.g. 256-bit) expands into $O(nR)$ masks and scales, giving an effective key size far larger than typical block ciphers. For instance, with $n=8$, $R=3$, $\ell=2$, masks and scales total hundreds of bits. Guessing the seed by brute force is thus infeasible.

- **Vector Space and Linear Algebra**: Viewing VEINN as operations in $\mathbb{Z}_{2^{16}}^n$ permits use of algebraic cryptanalysis (e.g. solving systems of equations). However, each coupling layer introduces bilinear terms (the convolution mixes halves nonlinearly across coordinates), and the random scalings break linear structure. Without knowing the masks, even linear/ differential cryptanalysis is difficult. In effect, each round resembles a large random permutation of the vector space, similar to a well-mixed cipher.

- **Comparison to AES and Block Ciphers**: Traditional block ciphers like AES operate on 128-bit blocks using bytewise S-boxes. VEINN's block is also 128 bits ($n=8$ words) by default, but encryption uses 16-bit arithmetic on the whole block at once. The INN structure can yield large "round feedback" while being efficiently implementable with matrix/vector ops (fast on hardware). AES is well-studied and believed quantum-vulnerable only to Grover's algorithm (quadratic speedup, not fatal). VEINN's advantage is lattice-based hardness: quantum algorithms do not break LWE efficiently, whereas AES-128 is reduced to ~2^64 quantum effort by Grover. Thus VEINN can be tuned for higher (classical) key-equivalents if needed.

Performance and Parallelism

- **Fast Arithmetic**: Encryption mainly involves additions, multiplications mod 2^{16}, and convolution by an all-ones vector (which can be done by summing with sign changes, or with small FFT optimizations if n is large). These operations are constant-time and efficient on modern CPUs/ GPUs (e.g. with vector instructions). There is no exponentiation or large integer arithmetic, unlike RSA or EC.

- **Parallel Blocks**: Each block encryption is independent (except for HMAC). The invertible network's layers can also be parallelized across coordinates. Thus VEINN has a high parallelism potential, similar to stream ciphers.

- **Homomorphic Sum/Product**: Homomorphic operations are simple ($+$ and convolution) on vectors, also efficient. This makes VEINN appealing for applications where such operations are frequent (e.g. secure aggregation, privacy-preserving ML). The absence of bootstrapping overhead (unlike FHE) is an advantage for limited computations.

Integration with Homomorphic Encryption

VEINN’s homomorphic capabilities mirror those of lattice-FHE in that ciphertexts form a ring. In particular, adding two VEINN ciphertexts corresponds to adding plaintext vectors in $\mathbb{Z}_{2^{16}}^n$, and their negacyclic convolution corresponds to polynomial multiplication. This is similar in spirit to BFV/CKKS schemes where ciphertexts in $R_q = \mathbb{Z}_q[x]/(x^n+1)$ can be added/multiplied. However, because VEINN is symmetric-key, it avoids public-key overhead (no large key or relinearization), at the expense of requiring a shared seed. One could combine VEINN with RSA-KEM for public-key use (the implementation hints at RSA for key delivery). Compared to CKKS [1], VEINN does not do approximate real arithmetic or rescaling, but it is simpler and faster for integer operations.

Cryptanalysis

We discuss potential attacks in the context of classical and quantum adversaries.

Classical Attacks

- **Brute-Force Key Search**: If the seed space is 2^k, a brute-force attack takes $O(2^k)$ symmetric operations. VEINN can choose $k\ge 256$ bits for security. Grover search would cut this to $O(2^{k/2})$ quantum time, requiring $k\ge 256$ to be safe (against 2^{128} effort, a typical PQ target).

- **Differential/Linear Analysis**: The coupling layers with modular convolution and odd scaling provide complex mixing. Each round is an invertible linear-affine transform $x \mapsto A_r x + b_r$ over $\mathbb{Z}_{2^{16}}^n$ (where A_r includes the coupling and scaling). Stacked rounds give $x \mapsto A x + b$ (overall), but the structure of A is hidden by random masks. Without knowledge of masks and scalings, an adversary sees only random-looking mappings. Known-plaintext attacks would require solving a large system of modular equations: given (x,y) pairs, find seed s such that $\text{permute_forward}(x; s) = y$. This is akin to solving a nonlinear system over a ring of composite modulus (2^{16}). No efficient algebraic attack is known; at best one could attempt multivariate algebra or SAT solvers on the bit-level equations, but the dimension ($n\approx 8$) is small and structure is high-degree.

- **Lattice Attacks on LWE**: If LWE-PRF is used, an adversary might try to recover the PRF seed by solving LWE: given many mask values (the *output* of the LWE-PRF) and knowledge of the PRF structure, solve for the secret vector s. In our scheme, the PRF uses a *fixed* secret s and a varying public a each invocation, plus small error e [5]. This is essentially RLWE: $b = a * s + e \pmod{Q}$ under negacyclic convolution. Breaking this requires lattice reduction in dimension n with modulus 2^{16}. Standard lattice attacks (BKZ, dual attacks) are expected to be computationally infeasible for reasonably chosen n (e.g. $n\ge 8$) and error distribution (ternary). No known classical algorithm solves general LWE in polynomial time. Thus LWE-PRF should resist classical cryptanalysis apart from exhaustive search in the large search space.

- **Side-Channel and Implementation Risks**: As with any cipher, side-channel (timing, power) could leak information. The code should be constant-time modulo operations and avoid secret-dependent memory access. The use of modular inverses on odd scalars must also be careful. These are generic concerns, not specific to VEINN design.

Quantum Attacks

- **Grover Search**: A generic quantum speed-up for symmetric ciphers. Grover's algorithm can find a preimage or key in $O(2^{k/2})$ time (with quantum queries to an oracle). To counter this, VEINN's seed length can be doubled (e.g. 512-bit seed for 256-bit post-quantum strength). The scheme itself uses only polynomial-time classical operations, so it is not inherently slowed by quantum adversaries except by brute force.

- **Quantum LWE Solvers**: No known *polynomial-time* quantum algorithms for LWE exist (unlike factoring or discrete log). There are heuristic subexponential algorithms for LWE on quantum (e.g. using quantum sampling, see Nigel Smart's blog [23]), but these do not break parameter sizes typical for secure FHE. If an attacker could break the LWE instances in the key schedule, they might recover masks and then invert the cipher by simple linear algebra. Thus VEINN's quantum security relies on LWE's presumed quantum hardness (as is the case for all RLWE schemes [2]).

- **Algebraic Attacks on Vector Operations**: Since VEINN's transforms are described by modular arithmetic, one might attempt quantum algorithms for solving modular equations (like solving short integer solutions via quantum lattice algorithms [24]). However, these are essentially variations of LWE/BKW which remain exponential in best cases. The architecture of VEINN introduces no new known quantum weaknesses beyond those of general LWE and multivariate integer equations.

Comparison to Existing Schemes

- **AES and Block Ciphers**: AES-128 has a 128-bit key and 128-bit block. It is extremely well-studied, with no known classical or quantum attacks aside from brute force (Grover's 2^{64}). VEINN can be configured for similar block size and effective key length, but with a fundamentally different primitive (algebraic vs. substitution). VEINN is not intended to *replace* AES in all contexts, but it provides similar functionality with a lattice-based security assumption and built-in homomorphism. Its invertible network structure also supports fast vector parallelism (multiple 16-bit words) which can be competitive with AES's SIMD implementation.

- **Lattice-based KEM/Encryption**: NIST PQC finalists (Kyber, CRYSTALS, NTRU, FrodoKEM, etc.) use LWE/RLWE for key exchange or public-key encryption, not direct symmetric encryption. These schemes have larger ciphertexts (KB-scale) and are public-key, whereas VEINN is symmetric (seed is shared via secure channel) and yields ciphertexts similar in size to plaintext (up to padding). VEINN could serve as a symmetric complement: e.g. use RSA or Kyber to share a session key, then VEINN for data encryption. VEINN's novelty is embedding lattice hardness in a cipher, not just KEM.

- **FHE (CKKS, BFV)**: VEINN is *partially homomorphic* (supports some homomorphism) but not fully (no bootstrapping). CKKS can perform arbitrary circuits up to a depth by relinearization. VEINN can add/multiply once or twice without error accumulation, but since it does not encrypt real values, its homomorphism is more limited (integer or ring operations only). On the other hand, VEINN's operations are much faster (no big polynomial arithmetic) and it does not expose noise growth issues: its "noise" (PRF randomness) is independent of operations. Thus VEINN may be suited for lighter homomorphic tasks or MPC.

Conclusion

We have detailed **VEINN**, a block cipher built from invertible neural network concepts over a modular vector space, with lattice-based key derivation for post-quantum security. By treating data and keys as vectors in $ \mathbb{Z}_{2^{16}}^n$, VEINN achieves fast, parallelizable encryption that supports homomorphic addition and multiplication. Its security derives from a mix of classical cipher complexity (invertible flow networks) and LWE hardness [2]. Preliminary analysis suggests that without breakthroughs in LWE or multivariate modular algebra, VEINN can resist both classical and quantum attacks. Compared to AES or ring-based schemes like CKKS, VEINN offers a unique trade-off: hardware-friendly arithmetic with built-in lattice entropy. While experimental, VEINN shows a promising direction for symmetric encryption in the PQC era, bridging cryptography and machine learning concepts.

Cryptanalysis Outlook. Detailed cryptanalysis remains future work. Potential attacks include differential and algebraic cryptanalysis of the invertible network, as well as lattice reduction on the embedded LWE. Initial estimates indicate that, with adequate parameter sizing (e.g. $n\ge8$, rounds ≥3, 128+ bit seed), exhaustive search and known quantum algorithms would be infeasible. However, VEINN's novel structure invites further study: proving rigorous security bounds or finding weaknesses in the coupling/scale layers will be important. Ultimately, VEINN stands as a conceptual fusion of block ciphers, neural invertibility, and lattice cryptography, meriting deeper analysis for practical deployment in post-quantum cryptography.

---

[1] [2] arxiv.org
https://arxiv.org/pdf/2205.03511

[3] [6] [7] Normalizing Flows with Real NVP | Bounded Rationality
https://bjlkeng.io/posts/normalizing-flows-with-real-nvp/

[4] [5] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] veinn.py
file://file-Ezjtm3R7W1QqxYdxpKepAa

[23] Implications of the Proposed Quantum Attack on LWE - Nigel Smart
https://nigelsmart.github.io/LWE.html

[24] [2310.00644] LWE with Quantum Amplitudes: Algorithm, Hardness ...
https://arxiv.org/abs/2310.00644