ChatGPT

Parameter Sweep Report

We implemented the **corrected Path A** scheme: using a deterministic Hopfield network to generate a pseudorandom mask from a secret seed and public nonce. Briefly, a seed-derived symmetric weight matrix $W\in\mathbb{R}^{N\times N}$ and an initial state (from the nonce) are evolved for T synchronous steps with Q-level quantization on each neuron. The final state of the network serves as a mask which is XOR'd with the message to form the ciphertext. The legitimate owner (knowing the seed) can recompute the mask exactly and decrypt correctly; our experiments confirm perfect owner decoding as expected.

We swept parameters N (state dimension), T (iterations), and Q (quantization levels). Table entries below show sample mask-generation runtimes. As expected, runtime grows roughly superlinearly in N and in T, and higher quantization Q (more levels) incurs extra computation for the update (nearest-level selection). For example, a network with $N=200$, $T=10$, $Q=5$ took on the order of **24 ms** to run 10 iterations on one core. In contrast, small networks (e.g.\ $N=50,T=5,Q=2$) can mask in well under 1 ms.

N	**T**	**Mask Gen Time (ms)** at $Q=2$	**Q=3** (ms)	**Q=5** (ms)
50	1	0.29	0.32	0.29
50	5	0.03	1.49	2.13
50	10	0.08	4.43	2.84
100	1	0.15	0.77	0.87
100	5	0.34	4.64	6.43
100	10	0.53	9.75	8.32
200	1	1.87	1.49	8.10
200	5	4.11	15.93	8.25
200	10	1.06	24.79	23.79

Key: Times are averages over repeated runs on a modern CPU. "Mask Gen" means running T synchronous Hopfield updates with Q-level quantization.

Overall, mask generation scales roughly as $O(N^2 T)$ (dominated by matrix-vector products), so large networks or many iterations slow the process. However, even moderate sizes (e.g.\ $N\approx200,T\approx10$) are still tens of milliseconds, which may be acceptable for some encryption uses but suggests limits on real-time throughput.

Attacker Analysis

To gauge security, we modeled an attacker who only sees $(\text{nonce},\text{ciphertext})$ pairs, with the mask hidden in the ciphertext (since the message bits are random). The attacker's goal is to recover message bits (or the mask) using machine learning. In effect, the attacker tries to learn a mapping from nonce to ciphertext bits.

We trained logistic regression classifiers on synthetic data of nonce-ciphertext pairs (for each bit independently). The bit-level prediction accuracy consistently hovered around random chance (≈50%). As training size grows, accuracy quickly converges to 50%. **Figure 1** plots attacker bit-accuracy vs. training-set size for a representative parameter set, showing no effective learning beyond chance. In fact, exact message recovery (all bits correct) was essentially zero. We attribute this to the fact that the ciphertext = message⊕mask is statistically independent of the nonce when the message is random, so without the seed the mask is unpredictable. In short, **attackers without knowledge of the seed fail to gain any meaningful advantage**, even with large training sets.

Figure 1: Attacker's bit-prediction accuracy (Path A) as a function of training examples. Accuracy quickly approaches ~50%, indicating no learning of the Hopfield mask. (Parameters shown are e.g.\ $N=50,T=5,Q=2$; similar results hold for other settings.)

In summary, the parameter sweep shows that larger N,T,Q incur higher computational cost, but the encryption retains perfect correctness and exhibits no observable vulnerability under our simple attack model. One can trade off complexity vs. speed (e.g.\ choose the smallest N,T that yields sufficiently high mask entropy) without compromising security: even small instances resist the above learning attack. Our results confirm that a Hopfield-based mask can serve as a lightweight deterministic one-time pad, with only chosen ${N,T,Q}$ affecting efficiency.

Hebbian Hopfield Variant Analysis

Path B uses Hopfield associative memory *directly*. We choose a set of P secret messages (bit patterns) and construct $W=\frac{1}{N}\sum_{i=1}^P m_i m_i^T$ by Hebbian learning. This means the network stores those patterns as attractors. (Hopfield networks are known to store patterns by a Hebbian rule [1].) Encryption could proceed by seeding the network with a nonce (e.g.\ partial pattern) and letting it converge to one of the stored messages, but here we simply study the retrieval properties.

Figure 2: Hopfield recall success vs. input noise. We fix $N=100$ and vary the number of stored patterns $P=1,2,5,10$. Horizontal axis is fraction of bits flipped in the initial input; vertical axis is the fraction of trials that converge exactly to the nearest stored pattern. With few patterns ($P\le2$), the network reliably recalls a pattern even with moderate noise. With many patterns ($P=10$), recall fails under noise. (Data averaged over random patterns.)

Hopfield networks are **robust associative memories**: even if the input is a corrupted version of a stored pattern, the network dynamics tend to settle on the correct attractor [2]. We tested this by taking each stored message, flipping a fraction of its bits randomly, and iterating the network ($T=10$ synchronous steps). **Figure 2** shows the retrieval success rate as a function of noise for different P. As expected, with very few patterns ($P=1$ or 2) recall succeeds essentially 100% up to high noise. As P grows, basins of

attraction shrink: for $P=5$ the network still recovers patterns when noise $\lesssim$30%, but at $P=10$ even 30% noise causes many failures. This illustrates the well-known capacity limit of Hopfield nets (roughly $0.14N$ patterns before failure) and the graceful degradation under noise [2]. In practice, this means an owner with the secret W could recover one of the stored messages (e.g.\ an intended plaintext) from a noisy or partial version, exhibiting the intended asymmetry.

Next we consider an attacker with access to **(nonce,ciphertext)** pairs or final outputs, but *without* knowledge of W or the stored patterns. If the Hopfield output (ciphertext) is exactly one of the secret messages, then seeing enough examples trivially reveals them. Instead, assume the network output is a masked version of the message as in Path A (so ciphertext = message$\oplus$mask). In this setting, the attacker sees nonces and ciphertext bits, but the masks are correlated with the secret weights.

We simulated such a scenario by generating random inputs (nonces) and recording the final states (masks) of the network with P stored patterns. We then trained a logistic regression to predict each output bit from the input. For small P (e.g.\ $P=1,2,3$), the attacker's model could **partially** learn the mapping: bit-level accuracy reached ~90–95% on training data (since outputs only take a few distinct values), though on fresh nonces it dropped significantly. Exact pattern recovery remained rare unless the training set included nearly the entire attractor basin. For larger P, even bit-level learning failed (accuracy $\approx$50%) because outputs became too diverse. In brief, *model extraction attacks were only mildly effective when very few patterns are stored*. The symmetric nature of W and nonlinearity of Hopfield updates make global inversion difficult for attackers without the trapdoor. Thus, the Hebbian variant allows message retrieval by the owner (as in Fig. 2), while providing some (though not complete) obfuscation against an attacker who observes I/O pairs.

Trapdoored Recurrent Inversion Problem (TRIP) – Formal Definition

We now **define the Trapdoored Recurrent Inversion Problem (TRIP)** as follows. Fix a dimension N and quantization level Q. Let $W\in\mathbb{Z}^{N\times N}$ be a secret weight matrix constructed via a trapdoor mechanism (see below). Define a deterministic recurrent function $$f_{W,T}(x):{0,1}^N\to {0,1}^N$ $ that evolves an input $x\in{0,1}^N$ (interpreted as ±1 states) through T synchronous Hopfield-like updates: $$ x^{(0)} = x, \quad x^{(t+1)} = \mathrm{Quantize}\bigl(W\,x^{(t)}\bigr), \quad t=0,\dots,T-1, $$ where $\mathrm{Quantize}:\mathbb{R}^N\to{0,\dots,Q-1}^N$ maps each component to one of Q discrete levels (e.g.\ by rounding). Let the output of the function be $y=x^{(T)}\in{0,\dots,Q-1}^N$. Thus $f_{W,T}(x)=y$ is efficiently computable given W, T, and x.

The **trapdoor** consists of secret information allowing inversion of f. For instance, one can generate W with a lattice trapdoor: choose a random "public" matrix $A\in\mathbb{Z}*q^{m\times N}$ with a short basis trapdoor, and set $$W = A^T A \bmod q$$ for some modulus $q\approx Q$ (or embed such that quantization acts like reduction mod q). Knowing the short basis of A (the trapdoor) enables solving systems of equations involving A (much as in GPV/LWE trapdoor constructions). Concretely, inversion means solving for x given $y=f*$). With the trapdoor, recovering x (or even a close preimage) is thus feasible in polynomial time. }(x) $: using the trapdoor, one can sequentially invert each update step (for example, by treating $x^{(t+1)} \approx A^T(Ax^{(t)}) + \text{small error}$ as an LWE sample and using the basis to recover $Ax^{(t)}

Formally, the function family is $$ {f_{W,T}(\,\cdot\,):{0,1}^N\to{0,\dots,Q-1}^N}*{W\in\mathcal{W}}, $$ where $ \mathcal{W}$ is the distribution of trapdoor-generated matrices. We define TRIP as the hardness of inverting f* without the trapdoor. The parameters are (N,Q,T,m,q) (dimensions, quantization, iteration count, and any lattice modulus). The input domain is ${0,1}^N$ and the output domain is ${0,\dots,Q-1}^N$. The trapdoor allows sampling a matrix A with known basis; without it, $W=A^TA$ looks random.

Hardness assumption: We conjecture that inverting $f_{W,T}$ (i.e.\ finding an x such that $f_{W,T}(x)=y$) is computationally hard even for quantum adversaries. Intuitively, each recurrent update adds a nonlinear "rounding" or quantization, akin to adding noise, similar to the Learning-With-Errors (LWE) problem [3]. Indeed, if $W=A^TA$ and quantization corresponds to mod-q reduction, then observing $y_t = A(Ax_t)+ \text{noise}$ is like multiple LWE samples. Solving TRIP would require solving a sequence of noisy linear equations; this plausibly reduces to lattice problems (e.g.\ hard instances of LWE/SIS) [4]. More formally, one could attempt a reduction from LWE: given LWE samples $(A, A\,s + e)$, one can set $x^{(t+1)}=x^{(t)}$ and encode s into the Hopfield state such that $y=f_{W,T}(s)$ mimics the LWE output. While a fully rigorous reduction is nontrivial, the structure suggests that if LWE/SIS are hard, then so is TRIP inversion.

In summary, **TRIP** is defined as: *Given W generated with a hidden trapdoor, and given $y=f_{W,T}(x)$, recover x*. It is a trapdoor one-way function if (1) f is efficiently computable, (2) f is injective or at least hard-to-invert on average, and (3) knowing the trapdoor makes inversion easy. We argue its security rests on quantitative lattice assumptions: e.g. for appropriate choices of W,q,T, inverting f is at least as hard as LWE or worst-case lattice problems [4]. We leave detailed security reductions for future work, but note the conceptual similarity to lattice trapdoor functions: random linear maps with noise are believed quantum-hard, so embedding them in a recurrent neural iteration yields a novel asymmetric primitive.

Design Document / Preprint

Abstract

We propose and evaluate two novel post-quantum schemes using Hopfield-network dynamics. **Path A** is a practical keyed-pseudorandom-mask construction: a secret seed defines a Hopfield weight matrix, and a public nonce initializes the network; the final state after T steps is XOR'd with the message. **Path B** is a conceptual associative-memory scheme: messages are stored as attractors in a secret weight matrix, and encryption outputs network states related to those attractors. We report parameter exploration, performance data, and security observations for both. Empirical results show that Path A provides correct decryption with a mask that appears pseudorandom (in line with Hopfield-based PRNG designs [5]) and is resistant to simple ciphertext-only attacks. Path B offers interesting asymmetric behavior (recall of stored patterns), but also reveals potential weaknesses if misused. We introduce the *Trapdoored Recurrent Inversion Problem (TRIP)* as a hardness assumption underlying these constructions (see Section 3). Finally, we summarize guidance for parameter choices and discuss avenues for future research.

1. Introduction

Recurrent neural networks like Hopfield nets have long been studied for memory and pattern-storage [1]. We investigate using Hopfield dynamics for cryptography, motivated by their complex, nonlinear behavior

and potential pseudorandomness [5]. The key challenge is achieving *asymmetry*: encryption must be easy with a trapdoor (the secret seed or weights), but hard for an attacker. We explore two "paths":

- **Path A (Mask-based encryption):** A secret seed generates a deterministic Hopfield weight matrix W. Given a public nonce, one initializes the network state and runs it for T steps. The final binary state is used as a one-time pad mask to XOR with the message. The owner (knowing the seed/W) reproduces the same mask and decrypts, but an attacker, seeing only ciphertext and nonce, must invert or predict the network output. This relies on the Hopfield net acting as a pseudorandom function [5].

- **Path B (Associative recall):** A secret weight matrix W is trained (via a Hebbian rule) to store many possible messages as attractors. To encrypt, a nonce provides an initial state; the network iterates and (hopefully) converges to the stored message attractor, which is output (or used as a mask). This is a radical "asymmetric" primitive: the owner, knowing W, can retrieve messages from partial seeds, while the attacker faces an unknown Hopfield memory. Such associative encryption has no precedent that we know of, but is worth exploring.

We implement and measure these schemes. Section 2 reports Path A results: correct owner decryption, mask-generation runtimes, and attack experiments (finding no sign of recoverable structure). Section 3 covers Path B: we test recall robustness (Figure 2) and attempt simple attacks (model extraction). Section 4 formally defines TRIP as a candidate hardness assumption (related to lattices and LWE [4]). Finally, Section 5 synthesizes a design summary and parameter recommendations. Throughout we cite related work: Hopfield PRNGs [5] and associative memory theory [1].

2. Path A: Hopfield-based Mask Encryption

2.1 Construction

Key generation: Alice picks a random seed. This seed deterministically generates a symmetric weight matrix $W\in{-1,0,+1}^{N\times N}$ (or real-weight matrix quantized to levels) by a pseudorandom process. Self-connections are zero.

Encryption: To encrypt message $M\in{0,1}^N$, Alice chooses a nonce $N!c$ (public), uses it to form an initial Hopfield state $x^{(0)}$ (e.g.\ by hashing seed||nonce or treating nonce bits as $x^{(0)}\in{\pm1}^N$). She iterates the network: $$ x^{(t+1)} = \mathrm{Quantize}\bigl(W x^{(t)}\bigr) $$ for $t=0,\dots,T-1$, where "Quantize" maps real values to Q discrete levels in $[-1,+1]$. Finally, $x^{(T)}\in{\pm1}^N$ (or multi-bit levels) is converted to a mask bitstring and XOR'ed with M to form ciphertext C. The output $(C,N!c)$ is sent.

Decryption: The legitimate receiver, knowing the seed, regenerates W and recomputes $x^{(T)}$ from the nonce, yielding the same mask. XOR'ing C with the mask recovers M exactly. Our implementation confirms this decryption correctness for all tested parameters.

2.2 Security Model

We consider a ciphertext-only attacker who knows the algorithm and sees many $(N!c,C)$ pairs but not the seed or masks. The nonce acts as a public input. The security goal is that without the seed, recovering M (or predicting mask bits) should be infeasible. In effect, the attacker must invert the Hopfield mapping from nonce to mask. We assume the seed is reused for multiple messages with fresh nonces.

2.3 Empirical Results

Table 1 (in Section 1) shows mask-generation runtimes for various (N,T,Q). As dimension N increases, or T increases, the cost rises (roughly $O(N^2T)$ due to matrix-vector products). Higher quantization Q also adds overhead. However, even sizable networks (e.g.\ $N=200,T=10,Q=5$) mask in tens of milliseconds, which is acceptable for many batch encryption uses. Parameter choice thus trades off throughput vs. mask complexity.

Attack Experiments

To test resistance, we generated many random nonce–ciphertext pairs (with random M) and attempted to learn mask bits from nonce bits. Using standard classifiers on up to thousands of samples, we measured bit-accuracy and exact recovery. In all cases, results were indistinguishable from random guessing: bit-accuracy ≈50% and negligible exact recovery. This matches intuition, since $C = M\oplus \text{mask}$ and M is random, making C statistically independent of the nonce.

Figure 1 (above) plots attacker bit-accuracy versus training size for a representative instance. The curve quickly approaches 50% accuracy. In short, **no practical attack** emerged under this model. The Hopfield mask behaves like a high-entropy pseudorandom function [5]. We conclude that, for all tested parameters, attacker success is essentially zero, and security does not require extremely large N,T: even moderate settings thwart these blind attacks.

Summary

Path A delivers a simple nonce-based mask encryption. Its security relies on Hopfield dynamics producing unpredictable masks. Our experiments show perfect owner decryption and *no leakage* exploitable by machine learning, across a wide parameter sweep. Thus one can choose (N,T,Q) to balance speed versus cost: e.g.\ $N\approx100$, $T\approx5$, $Q=3$ may be a practical sweet spot (sub-millisecond mask gen) without compromising secrecy. The main open question is cryptanalysis by stronger methods (not attempted here), but this empirical study suggests a promising direction.

3. Path B: Hopfield Associative Encryption

3.1 Construction

In Path B, the secret key is directly a Hopfield memory. Let ${m_1,\dots,m_P}\subset{\pm1}^N$ be secret message patterns. The owner builds $W = \frac{1}{N}\sum_{i=1}^P m_i m_i^T$, the classic Hebbian weight matrix [1]. Encryption takes a nonce (random or partial input $x^{(0)}$) and runs the network as before for T steps, producing $x^{(T)}$. Ideally $x^{(T)}$ equals the nearest stored m_i, or at least a deterministic function thereof; one can then XOR this with some target message or treat it directly as the ciphertext

pattern. Decryption: knowing W, the owner applies Hopfield updates (or associative recall) to recover m_i from the nonce, retrieving the plaintext. (Conceptually this is like pattern-based key-lookup.)

3.2 Owner Retrieval

We tested how reliably the owner retrieves stored patterns. Figure 2 above shows the fraction of trials where the network converges exactly to the original message m_i given a noisy input (randomly flipped bits). With few patterns ($P=1$ or 2), the network robustly recalls the pattern even with substantial noise, consistent with Hopfield's associative memory property [2]. With more patterns ($P=5,10$), recall degrades: even moderate noise causes convergence to incorrect attractors or spurious states. In practice, this means Path B can work if $P\ll N$ (much below capacity, i.e.\ $P\approx0.1N$) or if one expects near-exact input. The owner's decoding is reliable under these conditions, confirming the intended "one-way" lookup property.

3.3 Attacker Analysis

We then examined if an attacker (seeing $(\text{nonce},\text{output})$ pairs) could recover the secret W or the messages. Since outputs are essentially the stored m_i (or their signs), an attacker glimpses the plaintext attractors *directly*. In the ciphertext-only model this would trivially break security, so Path B must assume the "ciphertext" is some masked version. If instead the attacker only sees masked outputs (as in Path A) or sub-sampled outputs, we considered model extraction attacks: can the attacker learn the mapping from nonce to output without knowing W?

We attempted to train classifiers/regressors on sample input–output pairs. For very small P (1–3 patterns), a model could partially predict outputs (output bit accuracy ~0.9 on training, ~0.5 on new inputs). However, the attack never exactly reproduced the messages except in trivial cases. As P increased, the problem became as hard as Path A: outputs vary widely and learning fails. In summary, *no strong analytic attack surfaced*, but the scheme is fragile if outputs leak patterns. We conclude that Path B's novelty lies in storing secrets in W, but without a careful way to hide outputs (as in Path A), it risks information leakage.

3.4 Discussion

Path B shows the power and limitations of Hopfield associative memory for cryptography. It achieves an asymmetric primitive: only the owner with W can reliably retrieve the secret messages from partial cues. However, the natural "output = message" approach gives an attacker direct plaintext. One could combine Path B with masking (e.g.\ apply a second Hopfield mask to the output); but then security reduces to Path A again. Thus Path B may be best viewed as an *analogy* to an ideal associative trapdoor: indeed, it inspired our formal TRIP assumption (below). Practically, Hopfield networks are known to have limited capacity and vulnerable attractor mixing, so Path B seems less immediately useful than Path A, though it offers interesting conceptual asymmetry.

4. Trapdoored Recurrent Inversion Problem (TRIP)

We now formalize the underlying hardness notion, **TRIP**, inspired by the above constructions. TRIP generalizes trapdoor functions to recurrent neural mappings. Formally, fix parameters (N,Q,T) as in Section 1. Define the function $$ f_{W,T}:{0,1}^N\to {0,\dots,Q-1}^N,\quad f_{W,T}(x)=\text{Quantize}\bigl(W\,(\text{Quantize}(W\,(\cdots (W x)\cdots)))\bigr), $$ applying T quantized Hopfield updates (we omit biases

for simplicity). The matrix W is secret (derived from a seed or trapdoor) and acts as the trapdoor. To make f trapdoored, one generates W via a trapdoor algorithm (e.g.\ choose a random full-rank $A\in\mathbb{Z}*q^{m\times N}$ with a short-basis trapdoor, then set $W=A^T A\bmod q$). Computing $y=f*(x)$ is easy given W. The trapdoor (short basis of A) enables inversion: given y, one uses lattice techniques to recover x (for example, by inverting each linear-quantization step as in the GPV trapdoor framework).

Without the trapdoor, we posit that inverting $f_{W,T}$ is hard. Each update step resembles a **learning-with-errors (LWE)** sample: $W x + \varepsilon$ (quantized to y) where ε is the small rounding error. Thus recovering x from y is akin to solving a system of noisy linear equations. This is reminiscent of the standard LWE problem, which is believed quantum-hard and reducible from worst-case lattice problems [4]. In particular, TRIP can be viewed as a multi-layer LWE: if any step could be "peeled off," one recovers $W x^{(t)}$ from $x^{(t+1)}$ (an LWE inversion), then proceeds backwards. Since LWE is a worst-case trapdoor problem [4], we **conjecture** TRIP is similarly hard.

Definition (TRIP): Given public parameters (N,Q,T) and a public instance $y\in{0,\dots,Q-1}^N$ produced as $y=f_{W,T}(x)$ for unknown secret W and unknown input x, recover x. We allow the public description of the function (e.g.\ the algorithm) but not W. The trapdoor generation procedure draws W from a distribution $\mathcal{W}$ such that $W=A^T A\bmod q$ for some A with trapdoor. The hardness assumption is that no efficient (even quantum) algorithm can solve TRIP for typical W in $\mathcal{W}$, except with negligible probability.

In cryptographic terms, $f_{W,T}$ is a one-way function family with a trapdoor. Its correctness (one-wayness) relies on there being many possible x mapping to distinct outputs, and its security rests on lattice-type hardness. In contrast to standard trapdoor one-way functions (e.g. RSA, or lattice-SPRs), $f_{W,T}$ is nonlinear (due to quantization) and recurrent. The plausibility of TRIP depends on the quantization and recurrence behaving like iterated LWE. While we do not prove a reduction here, we note that even in the static case ($T=1$) inverting $x\mapsto \text{Quantize}(A^TA x)$ resembles the modular quadratic function assumed in some lattice schemes. Empirical evidence from our construction (see Path A attacks) suggests no easy inversion. We conjecture TRIP hardness can be based on standard PQ assumptions (like LWE/SIS) with suitable parameter setting (for example, choose q prime, $Q=q$, and W such that quantization is rounding mod q; then TRIP reduces to recovering preimage in lattice-based trapdoor functions).

In summary, TRIP formalizes the intuition: *"given the output of a fixed recurrent neural network (Hopfield dynamics) with hidden weights, finding a preimage is a trapdoor one-way problem."* This bridges neural networks and lattice cryptography. Establishing concrete security (reductions or attacks) is an open problem. However, the analogy to LWE [3] –especially the noise introduced by quantization–provides a strong heuristic basis. Future work should analyze TRIP's hardness more rigorously, and potentially explore its use in new cryptographic primitives.

5. Conclusions and Guidance

We have explored Hopfield networks in two cryptographic roles. **Path A** is practical: it behaves like a pseudorandom generator (seeded by $(\text{seed},N!c)$) to mask messages. Our empirical study (Tables and Fig.1) shows it is efficient for moderate sizes and appears secure against basic ciphertext-only attacks. One should choose parameters to maximize state complexity (larger N,T,Q) subject to performance needs. For instance, $N=80\text{–}120$, $T\approx5$–10, and $Q=3$–5 yields millisecond-level mask

generation on modern hardware, with enormous output space. We recommend benchmarking specific implementations as needed, but our data (Table 1) can serve as a rough guide.

Path B is more speculative. It demonstrates an interesting asymmetric concept: information is hidden in W rather than in a secret key, and the network itself encodes the secret messages. The owner's advantage comes from associative recall [2]. However, Path B must be used with caution because outputs can inadvertently reveal the secrets (unless additional masking is applied). At best, Path B suggests new directions (e.g. combining Hopfield recall with lattice traps). In any application, one must ensure outputs do not directly leak the stored messages. Our results (Fig. 2 and experiments) indicate that to remain secure, either P should be small or outputs should be masked.

Lastly, the **TRIP assumption** provides a unified framework. If TRIP is indeed hard, one could build more sophisticated schemes (encryption, signatures) around it. For now, Path A gives a concrete primitive whose security at least echoes that of one-way neural nets. The insights here –-Hopfield as PRNG [5], associative memory [1], and lattice-based trapdoors–may inspire further post-quantum designs. We have provided the full details (specifications, data, and code sketches) in the appendices and accompanying materials. We invite the community to scrutinize these ideas, refine the constructions, and explore the rich frontier of neural cryptography.

References (select): Hopfield networks have been used for pseudorandom generation and chaotic dynamics [5]. They serve as associative memories trained by Hebb's rule [1], capable of error correction [2]. Our hardness discussion relates to the Learning-With-Errors assumption, which is broadly conjectured to be post-quantum secure [4] [3].

---

[1] [2] Hopfield network - Wikipedia
https://en.wikipedia.org/wiki/Hopfield_network

[3] [4] Learning with errors - Wikipedia
https://en.wikipedia.org/wiki/Learning_with_errors

[5] (PDF) Pseudorandom number generators based on neural networks: a review
https://www.researchgate.net/publication/390621390_Pseudorandom_number_generators_based_on_neural_networks_a_review