Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 1205–1210
DOI: 10.15439/2023F9901
ISSN 2300-5963 ACSIS, Vol. 35

Postquantum symmetric cryptography inspired by neural networks

Wojciech Węgrzynek*, Paweł Topa†
Faculty of Computer Science, Electronics and Telecommunications,
AGH University of Kraków
Kraków, Poland
Email: *wegrzynek@student.agh.edu.pl,
†topa@agh.edu.pl

***Abstract*—We introduce a novel approach to postquantum symmetric encryption that allows us to modify and continue to use any encryption scheme. By composing the encryption and decryption functions with the evaluation of arbitrarily wide neural networks we are able to verify that anyone performing these functions has access to at least a certain amount of memory. Since the number of qubits in quantum computers has been relatively slow-growing, this provides us security from the Grover's search attack, and any attack utilizing a similar oracle circuit.**

***Index Terms*—post-quantum cryptography, symmetric key cryptography, encryption, neural networks**

I. Introduction

THE development of quantum computers is likely to lead to major breakthroughs in many areas of science and engineering. Cryptography is one area where this breakthrough is already evident. Although we do not yet have a quantum computer capable of cracking the 2048-bit RSA key, the world is preparing for that moment. In 2016, the US NIST announced a competition for a post-quantum public key algorithm. In June 2022, after three rounds of review, four algorithms implementing key encapsulation (Crystals-Kyber) and digital signature functionality (Crystals-Dilithium, Falcon, Sphincs+) were selected.

Secret key cryptography is much less threatened by quantum computers. The Grover algorithm is only able to halve the security strength of the AES algorithm. This means that AES with a 256-bit key will be as secure as AES with a 128-bit key is today.

This is still enough of a security margin not to change the cipher, which is the "workhorse" of the Internet, too quickly.

However, it is worth considering all possible directions for post-quantum secret key cryptography. Here, we present an idea of making Grover's search attack more challenging in terms of qubits of memory needed on a quantum computer.

II. State-of-the-art

Currently, the most widely used symmetric cipher is AES [1], which is susceptible to the Grover's search attack. AES exists in three variants: AES-128, AES-196, and AES-256 named after the length of the binary key string. The key spaces are then of size 2^{128}, 2^{196}, and 2^{256} respectively. Since Grover's search attack effectively halves the exponent of the key space size, AES-256 would be reduced to the security level of AES-128 and the remaining two variants would become insecure by the previous standard. Since AES-128 is currently considered secure, switching to AES-256 is the solution provided in [2] and [3]. Additionally, for purposes requiring AES-256 level security, [3] extends AES to include a 512 key length variant.

It is first worth noting that there already exist solutions that are secure against the Grover's search attack, in particular AES-QPP, which is a variant of AES in which the SubBytes and AddRoundKey are replaced by a Quantum Permutation Pad operation, granting quantum safety [4], or Saturnin, which is a block cipher that has been specifically designed for the purpose of being quantum-safe while also maintaining lightweight properties making it more suitable for IoT applications [5]. In comparison to them, however, AES has the benefit of having been exposed to extensive analysis by the public.

Using neural networks cryptographic solution, at least academically, is not a novel concept either. In [6] the authors utilize recurrent neural networks of a specific shape to define a symmetric cipher. In [7] a cryptographic hash function is defined based on the evaluation of a neural network with randomized matrices.

III. Preliminaries and Definitions

In this section, we introduce the terms and notation we will use throughout this article. Since this work exist at the intersection of a few different subfields within computer science, we divide it into subsections

A. Algebraic notation

For the sake of simplicity of notations in this subsection, we will introduce some shorthands that we will use throughout this article.

Let $v \in \mathcal{F}^n$ be an n-dimensional vector over some field $\mathcal{F}$, by $v[i]$ for $0 \leq i < n$ we will denote the i-th element of the vector.

A **Galois field** is synonymous to a finite field and we use the notation $GF(q)$ to denote a Galois field of size q. By a well-known algebraic theorem when q is of the form $q = p^n$ for some prime p and some $n \in \mathbb{N}_+$ then $GF(q)$ exists and

is unique in the sense of isomorphisms. For any other size a finite field does not exist, thus $GF(q)$ is well-defined only for powers of a prime number.

B. Cryptography

Here we attempt to formalize some notions present in cryptography. Note that it is particularly difficult to reflect the practical nature of this field. We nonetheless make the attempt in order to provide a mathematical argument for the correctness of our claims.

Definition 1. *A **symmetric cipher** is a tuple $(e, d, \mathcal{P}, \mathcal{C}, \mathcal{K})$ such that:*

- *$\mathcal{P}$ is a set of plaintexts,*
- *$\mathcal{C}$ is a set of ciphertexts,*
- *$\mathcal{K}$ is a key space,*
- *$e : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$,*
- *$d : \mathcal{C} \times \mathcal{K} \to \mathcal{P}$,*
- *for any $p \in \mathcal{P}$, $k \in \mathcal{K}$: $d(e(p, k), k) = p$.*

A cipher $\sigma = (e, d, \mathcal{P}, \mathcal{C}, \mathcal{K})$ is going to be **secure** if the following conditions are met:

1) Given only the value of $e(p, k)$ it is impossible to reliably compute p faster than the naive approach of iterating through all of the key space.
2) Given only the values of $e(p, k)$ and p it is impossible to reliably compute k faster than the naive approach of iterating through all of the key space.

Any algorithm or process that proves a cipher to be insecure is called an attack. Note that this is one place where there is a discrepancy between this formal definition and the practical notion of security — many ciphers are still considered practically secure despite existing attacks because those attacks are proved to be impractical.

An attack that violates the first property will be called a **ciphertext-only attack** and an attack that violates the second (but not the first) will be called a **known-plaintext attack**.

C. Neural networks

Since the main result of this work is heavily inspired by neural networks, we feel the need to define some notions from that field of study. Let us start with defining, arguably, the simplest type of neural network — the multilayer perceptron. This will be the only type of neural network we will refer to in this work, so we will sometimes use the term neural network as a synonym for a multilayer perceptron, but we note that in the wider topic such equivalence would be false.

Definition 2. *A **multilayer perceptron** over the field $\mathcal{F}$ is a function $f : \mathcal{F}^{n_1} \to \mathcal{F}^{n_{d+1}}$ of form $f = l_1 \circ l_2 \circ \cdots \circ l_d$, where $d \in \mathbb{N}_+$ is the **depth** of the neural network. Each function $l_i : \mathcal{F}^{n_i} \to \mathcal{F}^{n_{i+1}}$ for $1 \leq i \leq d$ (which we will call a **layer**) must be in the form $l_i = m_i \circ a_i$ where:*

- *$m_i : \mathcal{F}^{n_i} \to \mathcal{F}^{n_{i+1}}$ is a linear transformation using the matrix $M_i \in \mathcal{F}^{n_i \times n_{i+1}}$,*
- *$a_i : \mathcal{F}^{n_{i+1}} \to \mathcal{F}^{n_{i+1}}$ is a non-linear transformation (we will sometimes call a_i the **activation** function).*

Note that the typical definition is usually constrained to $\mathcal{F} = \mathbb{R}$ and also requires the activation function to be differentiable and monotonic on each element. This is a conscious choice on our part, since we need to generalize this concept to other fields, because of the impracticality of representing real numbers on computers.

D. Quantum computing

A **qubit** is the most fundamental unit of quantum information. The state of a qubit is any vector $\psi \in \mathbb{C}^2$ with its norm equal to 1. We traditionally denote the state of a qubit as a ket in bra-ket notation (also called Dirac notation), like so $|\psi\rangle$. Two special states, forming an orthogonal basis, are usually distinguished:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

If more than one qubit exists, the state of such a system is the Hadamard product of the states of each qubit. In bra-ket donation, the Hadamard product of two states $|\psi\rangle$ and $|\phi\rangle$ is denoted by $|\psi\rangle |\phi\rangle$, or sometimes $|\psi\phi\rangle$. For an n-qubit system we also define the orthogonal basis $\{|0\rangle_n, |1\rangle_n, \cdots, |2^n - 1\rangle_n\}$ where $|i\rangle_n$ is a vector such that $|i\rangle_n [i] = 1$.

A quantum gate O, acting on an n-qubit state, is any invertible, unitary $2^n \times 2^n$ matrix over complex numbers. The application of O on the state $|\phi\rangle$ is the product of the matrix and vector, and is denoted as $O |\psi\rangle$.

Notice that because of the definition of a quantum gate, to uniquely identify a quantum gate, it suffices to define the results of applying it to some basis. Below we use this fact to define some gates that will be referenced throughout this article.

The NOT gate, or the X gate is a quantum gate acting on a single qubit, and is defined as follows:

- $X |0\rangle = |1\rangle$,
- $X |1\rangle = |0\rangle$.

The Toffoli or the CCX gate is a quantum gate acting on a 3-qubit state. The Toffoli gate acts according to the following rules:

- $CCX |11a\rangle = |11\rangle (X |a\rangle)$,
- $CCX |abc\rangle = |abc\rangle$, if $|ab\rangle \neq |11\rangle$.

IV. Cryptanalysis with quantum computers

The development of quantum computing technology poses a threat to our existing, widely used, ciphers. In the field of public-key cryptography, there is, for example, the famous Shor's algorithm, the usage of which can break RSA (Diffie-Hellman, ElGamal and elliptic curve cryptography too) in polynomial time. For private-key encryption the known quantum attacks are much less spectacular, nonetheless, they do exist and are worth investigating.

Grover's search algorithm is a quantum computing algorithm that, given an oracle circuit Q, over $n + 1$ qubits, with

the property that for any $x \in \{0, 1, \cdots, N-1\}$, and for any $a \in \{0, 1\}$

$$Q\left|x\right\rangle\left|a\right\rangle = \begin{cases} \left|x\right\rangle X\left|a\right\rangle, \text{ iff } x = a \\ \left|x\right\rangle\left|a\right\rangle, \text{ otherwise} \end{cases}$$

is able to find a with only $O(\sqrt{N})$ invocations of Q. This is an obvious improvement over an analogous situation in classical computing where the fastest such algorithm needs $O(N)$ invocations.

For any symmetric cipher $\sigma = (e, d, \mathcal{P}, \mathcal{C}, \mathcal{K})$ we may then define the following known plaintext attack:

1) Define an oracle Q that given k_1 as input, returns $e(p, k) = e(p, k_1)$.
2) Use Grover's search algorithm to compute k.

This takes $O(\sqrt{|\mathcal{K}|})$ invocations of Q, making it an attack on σ. Note that this attack is only possible if the attacker is able to execute the oracle circuit — which is what we make use of in this work.

V. The Proposed Solution

We introduce a novel approach that allows one to secure any private key cipher against a Grover attack, provided that both communicating sides have more memory bits on their machines than an attacker might have on their quantum computer. The algorithm works by modifying a scheme σ to form a scheme σ', such that the decryption function σ' requires at least a set number of bits of memory to compute — thus preventing the formation of the Grover oracle. In this section, we will detail how we achieve this.

Let's start with an arbitrary encryption scheme $\sigma := (e, d, \mathcal{P}, \mathcal{C}, \mathcal{K})$ and a bijection $f : \mathcal{C}^n \to \mathcal{P}^n$, for some n, the details of which we outline in Section VI. The scheme $\sigma' := (e', d', \mathcal{P}^n, \mathcal{C}^n, \mathcal{K})$ is then defined such that for a key $k \in \mathcal{K}$:

- $e'((p_1, p_2, \cdots, p_n), k) := (e(f(e(p_1, k)), k), \cdots, e(f(e(p_n, k)), k))$,
- $d'((c_1, c_2, \cdots, c_n), k) := (d(f^{-1}(d(c_1, k)), k), \cdots, d(f^{-1}(d(c_n, k)), k))$.

In other words, to encrypt a message, we concatenate n messages encrypted with σ, pass the output through f, split it back into n messages, and encrypt them again. To decrypt a message we then: decrypt the ciphertexts using σ, concatenate them, pass the output through f^{-1}, split them, and decrypt them using σ.

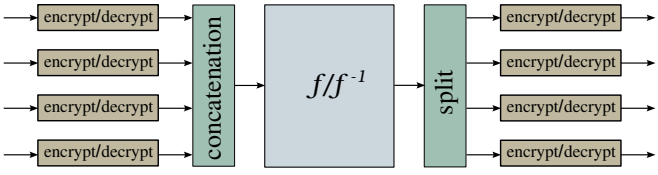

Figure 1: The proposed solution: securing cipher against Grover's attack with nonlinear bijective function f

Let us define a k-mixing function:

Definition 3. *A function $g : GF(2)^n \to GF(2)^n$ is k-**mixing** if, it is impossible to compute values for any subset of output bits, given only k of input bit values.*

Let us then state the following observation

Observation 1. *The proposed above solution produces a scheme that is secure against the Grover attack on machines with at most k qubits if the function f is k-mixing.*

Proof. Since the Grover attack requires the attacker to formulate an oracle circuit, and the attacker has at most k qubits available, this oracle function would have to compute values of some output bits given only k of the input bits, since that is all the attacker could store in memory. □

All that is now needed is to propose a family of bijective functions that will contain k-mixing functions for arbitrarily large k-s. At first glance, traditional multilayer perceptrons could be utilized. Indeed, a sigmoid activation function and invertible weight matrices would guarantee bijectivity, and the expressive power of neural networks should, at least intuitively, correlate to the mixing property, if random weights were used. However, with this approach, a practical issue arises - the naive implementation of such a function utilizing floating point arithmetic, would almost certainly not be able to produce an exact mathematical inverse. Instead, in the following section, we introduce a structure that operates on the finite field $GF(2)$ in a similar way that traditional multilayer perceptrons operate on $\mathbb{R}$, which we then may use instead.

VI. Definition of the f function

Let us define a Galois neural network:

Definition 4. *A **Galois neural network** is a multilayer perceptron over a Galois field.*

Let's observe that such a Galois network may serve as our f function, as long as we design it to be a bijection. To do so, it suffices to design a layer that is bijective. For the linear part of each layer, since we have total control of the values of the matrices, it suffices to find an invertible matrix M. For the activation function, this is less trivial, partially since, unlike traditional neural networks, all bijections over the field $GF(2)$ are linear, and so this activation function must in some way require interaction between elements of a vector. Below, we detail the design of the family of functions that fit these requirements.

We define a family of functions that work analogously to the iterative application of the Toffoli gate in quantum computers.

Definition 5. *A function $t_{m,k,l} : GF(2)^n \to GF(2)^n$ will be called a **Toffoli function** acting on bit m with control bits k, l (where m, k, l are pairwise different and $k < l$) if for all $v \in GF(2)^n$:*

1) $\forall_{i \neq m} t_{m,k,l}(v)[i] = v[i]$,
2) $(t_{m,k,l}(v)[m] \neq v[m]) \iff (v[k] = v[l] = 1)$.

The following lemma is then true:

Lemma 1. *Let $f : GF(2)^n \to GF(2)^n$ be a composition of at least one Toffoli function $f = t_{m_1,k_1,l_1} \circ t_{m_2,k_2,l_2} \circ \cdots \circ t_{m_i,k_i,l_i}$, such that for any $1 \leq \alpha, \beta \leq i$ where $\alpha \neq \beta$, at least one of the following is true:*

1) $m_\alpha \neq m_\beta$,
2) $k_\alpha \neq k_\beta$ *or* $l_\alpha \neq l_\beta$.

Then f is

1) *bijective,*
2) *nonlinear.*

Proof. The function is trivially bijective since $t_{m_1,k_1,l_1} = t^{-1}_{m_1,k_1,l_1}$.

Let us then prove nonlinearity, and suppose, by contradiction f is linear. There have to then exist $A \in GF(2)^{n\times n}, b \in GF(2)^n$ such that $f(v) = Av + b$ for any $v \in GF(2)^n$. Take t_{m_1,k_1,l_1} and consider a vector v_0, that is defined as follows:

- $v_0[i] = 0$ for all $i \notin \{k_1, l_1\}$,
- $v_0[i] = 1$ for all $i \in \{k_1, l_1\}$.

We know, from the definition of f that $v_0[m_1] = 1$, on the other hand $v_0[m_1] = (Av_0 + b)[m_1] = A[m_1]v_0 + b[m_1] = A[m_1][k_1] + A[m_1][l_1] + b[m_1]$. Notice that $b[m_1]$ must equal 0 since $f(0) = 0$, by definition of a Toffoli function. That means exactly one of $A[m_1][k_1], A[m_1][l_1]$ must equal one. Suppose, without loss of generality it is $A[m_1][k_1]$. Consider then a vector v_1, that only has a one as its k_1-th element. By definition of a Toffoli function, $f(v_1) = v_1$, but since $A[m_1][k_1] = 1$ and $b[k_1] = 0$, $f(v_1)[m_1] = 1 \neq v_1[m_1]$, thus we have a contradiction. □

Notice how Lemma 1 guarantees exactly the requirements for an activation function for a bijective Galois neural network.

We now formulate a conjecture which, if true, would guarantee security by Observation 1.

Conjecture 1. *For each $w \in \mathbb{N}_+$ there exists a w-mixing Galois neural network.*

We will try to argue for the validity of this conjecture by applying some statistical tests to randomly generated Galois neural networks in Section VIII.

VII. Performance Test

In this section, we evaluate the applicability of this approach through performance test results. We note, however, that the implementation used for these tests has likely yet to be fully optimized, one can expect enhancement in that regard with further development.

The Galois neural networks have been implemented using bitwise logical operations in Numpy [8] and the correctness of this approach was validated against the Galois package [9] for Python.

We will contain ourselves to single-layer NNs for the purposes of this evaluation, since increasing the number of layers since increasing the number of layers results in almost exactly linear growth of computation time.

To establish a frame of reference, we will contrast these results with the AES-256 implementation found in the PyCryptodome [10] package, note however that while we choose to compare to AES, because of its popularity, our solution is designed to work with any cipher, in particular less performant ones, where the performance gap might be less noticeable.

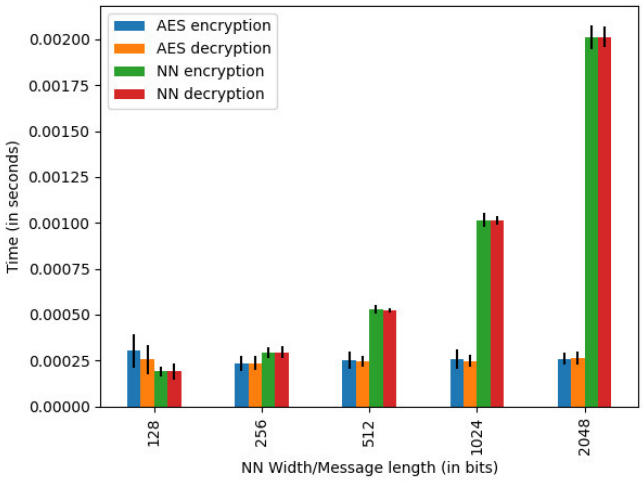

Figure 2: Results of performance tests. Encryption/decryption time contrasted with that of AES.

We compare the performance of a Galois neural network of a certain width on a single input with the performance of AES on that same input. For each of the lengths 128, 256, 512, 1024, and 2048 we ran 20 randomized messages, encrypted and decrypted them, and recorded the means and standard deviation. The tests were run on a mid-range laptop.

The results of these tests can be seen in Figure 2. Unsurprisingly, as Galois neural network evaluation has cubic complexity with respect to the width, the neural network execution time quickly trumps that of AES. However, this level of increased execution time might still be acceptable for applications where speed is not of high priority. Nonetheless, for most usecases, this data indicates a need for optimization, especially past the 512 mark.

Note that the results for AES do not seem to increase as the length of the messages increases. This is likely due to some kind of parallel execution.

VIII. Statistical Tests

To investigate the validity of Conjecture 1 we performed a suite of tests to examine some properties of Galois neural networks with randomized but constant weights.

A. Uniform distribution test

First, we propose the following test. Let w be the (even) width of a Galois neural network, randomly select the first $w/2$ bits of the input. Then repeatedly randomly select the remaining $w/2$ bits, concatenate all w bits together, compute the output of the network, and record the first few bits of the output. For a $w/2$-mixing function, we would expect a roughly uniform distribution of output values.

We performed this test, recording the counts for each output value, we then used the chi-squared test to assess how alike

size	depth	chi on 2 bits (3 degrees of freedom)	chi on 3 bits (7 degrees of freedom)	chi on 4 bits (15 degrees of freedom)	chi on 5 bits (31 degrees of freedom)
128	1	2.289062	6.421875	16.93750	36.2500
	2	1.929688	2.468750	16.56250	51.8125
	3	1.906250	7.984375	30.34375	33.5625
	4	3.375000	9.593750	6.28125	30.7500
256	1	2.781250	7.125000	7.18750	29.0625
	2	3.398438	15.875000	16.93750	25.5625
	3	1.117188	5.406250	7.93750	31.7500
	4	5.164062	4.843750	11.78125	19.7500
512	1	2.953125	3.125000	7.43750	25.7500
	2	2.843750	4.078125	18.71875	44.3125
	3	0.101562	6.562500	21.43750	24.4375
	4	4.507812	3.218750	9.06250	23.3125
1024	1	2.210938	8.859375	17.65625	30.2500
	2	1.890625	7.921875	10.18750	38.1875
	3	9.242188	9.468750	13.78125	38.2500
	4	3.226562	5.046875	14.56250	31.2500
2048	1	6.390625	6.343750	17.71875	39.0625
	2	2.039062	4.781250	20.90625	51.7500
	3	1.742188	9.062500	9.00000	26.7500
	4	1.882812	4.796875	18.68750	51.3125

Figure 3: χ^2 values from 1024 iterations of the uniform distribution tests.

these outputs are to those from a uniform distribution. The results values of those tests can be viewed in figure 3. The critical values for significance level 0.01 and the given degrees of freedom are (approx.) 11.345, 18.475, 30.578, 52.191 respectively. Thus all those tests failed to reject the null-hypothesis with a significance level of at least 0.01.

B. The bit flip test

Another property a mixing function should have, is that a small change in the input should have a large impact on the output. In [11] this property is tested in the following way:

- Consider x a random input to the function, record $f(x)$.
- Change a random bit of x, call it x', record $f(x')$.
- Compute the number of bits that are different between $f(x)$ and $f(x')$.

If f were a random permutation, we would expect the number of differing bits to follow a binomial distribution. Since we expect f to behave like a random permutation, we may use the Student's t-test to check if that is the case.

We performed 1000 samples of this test for every combination of $1, 2, 3$, and 4 deep and 128, 256, 512, 1024, and 2048 wide Galois neural networks. We then computed the Student's t test z statistic for each of those combinations and found the highest of these values to be less than 0.002, since for significance level 0.01 and 999 degrees of freedom, the passing z value is around 2.58, this indicates passing of the Student's t test.

C. Nonlinearity measurement

One of the fundamental properties of neural networks is their lack of linearity, in traditional applications this is useful because it lets them express and approximate nonlinear functions. Here, we would also benefit from a similar property, as if our function f were to be linear, it would be easily recognizable from a random permutation. We had already proven in Lemma 1 that, at least a single layer, Galois neural network would be nonlinear. In this section, we investigate the extent of nonlinearity, as depth increases.

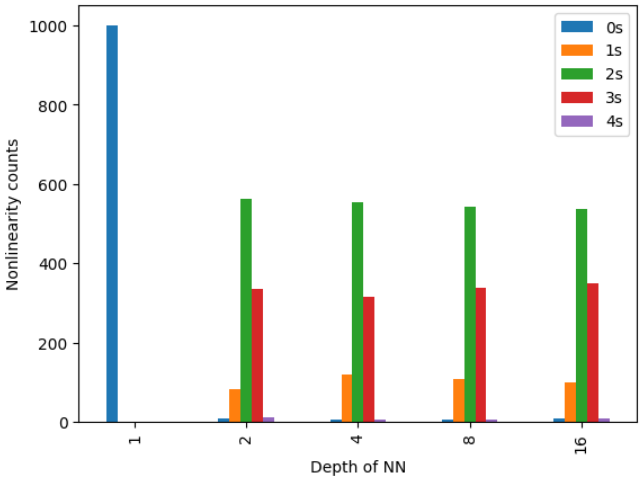

Figure 4: The distribution of nonlinearity for $w = 128$, $t = 5$, exponentially growing GNN depth. 1000 samples.

In [12] the nonlinearity of a function $f : \mathcal{F}^n \to \mathcal{F}^m$ where $m, n \in \mathbb{N}_+$, and $\mathcal{F}$ is a finite field, is defined in the following fashion.

Definition 6. *The **nonlinearity** $\mathcal{N}$ of a function $f : \mathcal{F}^n \to \mathcal{F}^m$ is given as:*

$$\mathcal{N}(f) = \min_{(u,w,v)\in\mathcal{F}^n\times\mathcal{F}^m\times\mathcal{F}} \#\{x \in \mathcal{F}^n | w^t f(x) \neq u^t x + v\}$$

In other words, we look at how closely we may approximate a projection of the output to a scalar, with an affine transformation of the input. We may use this definition to apply a measure to the notion of nonlinearity and express how nonlinear each depth of a neural network is.

Unfortunately, the naive computation of $\mathcal{N}(g)$, where $g : GF(2)^n \to GF(2)^m$ requires $\mathcal{O}(2^{n+m})$ time complexity, and $\mathcal{O}(2^n)$ invocations of g. This is too steep to compute for a wide Galois neural network. Instead, we may propose the following experiment:

1) Define a Galois neural network of width w, let $f : GF(2)^n \to GF(2)^n$ signify its computation.
2) Take a small number t.
3) Randomize the last $w - t$ bits and call them v_s.
4) Consider a function $f_t : GF(2)^t \to GF(2)^t$ that for input v_p is equal to $f(c(v_p, v_s))$ constrained to the first t bits, where c is a function that concatenates two vectors.
5) Compute the nonlinearity of f_t.

Such an experiment may help us derive some insights into the function f, and the efficiency of the activation function. Note that unlike measuring the nonlinearity of f, this is a random process so we will have to repeat it to gather reliable data.

In figures 4 and 5 we can see the results of such experiments. We may notice that for only one layer, and therefore only one activation function the nonlinearity values seem low and constant. This might be the result of the exact

depth	0	13	14	15	16	17	18	19	20
1	100	0	0	0	0	0	0	0	0
2	0	1	2	6	13	42	31	4	1
4	0	0	0	7	17	31	37	8	0
8	0	0	0	8	14	35	40	3	0
16	0	1	1	5	21	35	31	6	0

(a) $t = 6$

depth	32	36	37	38	39	40	41	42	43	44
1	100	0	0	0	0	0	0	0	0	0
2	0	4	0	8	0	38	0	45	0	5
4	0	1	2	7	10	17	25	28	10	0
8	0	1	2	3	12	22	20	26	13	1
16	0	1	1	4	8	23	24	18	16	5

(b) $t = 7$

depth	64	86	87	88	89	90	91	92	93	94	95	96	97
1	100	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	2	0	5	0	26	0	51	0	15	0
4	0	2	0	3	2	5	14	20	17	19	11	7	0
8	0	2	0	4	2	9	8	18	15	23	13	5	1
16	0	1	1	2	2	8	14	21	14	17	18	2	0

(c) $t = 8$

Figure 5: The distribution of nonlinearity values for larger t-s, $w = 128$, exponentially growing GNN depth. 100 samples for each.

implementation of the activation function, but nonetheless, this seems like a strong argument not to use such a low-depth GNN.

It is also noteworthy, that, with the exception of depth equal to 1, while the GNN depth increases exponentially it fails to have a significant impact on the magnitude of nonlinearity (at least in these experiments). However, the results for depth 2 still show some non-random behavior, like the results for t equal to 7 and 8 seemingly avoiding odd values. While a depth of 2 seems to be enough for the purposes of nonlinearity, practically it may be wise to air on the side of caution and recommend larger values.

IX. Future Work

It seems that the biggest area for improvement with this algorithm is performance. In the current form, it seems unlikely that the algorithm would see widespread use.

One idea to deal with that limitation would be to see if sparse matrices could be utilized to improve performance without loss of the statistical properties. One could also propose a different architecture for the underlying Galois neural network.

A different approach would be to seek improvements in the technical implementation of the algorithm, and either find improvements on the side of the algorithm or try to achieve better hardware support.

The second area for potential further work is trying to further verify and work towards proving Conjecture 1. One could for example try to replicate existing and to design new statistical tests perhaps on a bigger scale in order to show the guarantees required for widespread usage.

Acknowledgment

The research presented in this paper was realized with funds from the Polish Ministry of Science and Higher Education assigned to AGH University of Krakow.

References

[1] J. Daemen and V. Rijmen, “Aes proposal: Rijndael,” 1999.

[2] D. J. Bernstein and T. Lange, “Post-quantum cryptography,” *Nature*, vol. 549, no. 7671, pp. 188–194, Sep. 2017. doi: 10.1038/nature23461. [Online]. Available: https://doi.org/10.1038/nature23461

[3] X. Bogomolec, J. G. Underhill, and S. A. Kovac, “Towards post-quantum secure symmetric cryptography: A mathematical perspective,” 2019, https://eprint.iacr.org/2019/1208. [Online]. Available: https://eprint.iacr.org/2019/1208

[4] R. Kuang, D. Lou, A. He, and A. Conlon, “Quantum safe lightweight cryptography with quantum permutation pad,” *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, pp. 790–795, 2021.

[5] A. Canteaut, S. Duval, G. Leurent, M. Naya-Plasencia, L. Perrin, T. Pornin, and A. Schrottenloher, “Saturnin: a suite of lightweight symmetric algorithms for post-quantum security,” Mar. 2019, soumission à la compétition ”Lightweight Cryptography” du NIST. [Online]. Available: https://hal.inria.fr/hal-02436763

[6] M. Arvandi, S. Wu, A. Sadeghian, W. Melek, and I. Woungang, “Symmetric cipher design using recurrent neural networks,” in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 2006. doi: 10.1109/IJCNN.2006.246972 pp. 2039–2046.

[7] J. Tchórzewski and A. Byrski, “Performance of computing hash-codes with chaotically-trained artificial neural networks,” in *Computational Science – ICCS 2022*, D. Groen, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, Eds. Cham: Springer International Publishing, 2022. ISBN 978-3-031-08754-7 pp. 408–421.

[8] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. doi: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[9] M. Hostetter, “Galois: A performant NumPy extension for Galois fields,” 11 2020. [Online]. Available: https://github.com/mhostetter/galois

[10] “Pycryptodome,” https://www.pycryptodome.org/, accessed: 2023-05-22.

[11] J. Tchórzewski, “Application of artificial neural networks as hashing functions,” Ph.D. dissertation, AGH University of Technology, 2022.

[12] K. Nyberg, “On the construction of highly nonlinear permutations,” in *Advances in Cryptology - EUROCRYPT ’92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, ser. Lecture Notes in Computer Science, vol. 658. Springer, 1992. doi: 10.1007/3-540-47555-9_8 pp. 92–98.