Redes Neurais Artificiais Aplicadas à Geração e Troca de Chaves Criptográficas Binárias

Autor[1]

[1] Instituição
Endereço

email@domínio.com

Abstract. *With cryptography, is possible to make data unintelligible for those who do not have access to the planned rules. Cryptographic keys guarantee the security of the encrypted data regardless of the algorithm used. These keys are usually generated from pseudo-random values, requiring entropy, limiting their sizes by necessity performance, among other negative consequences. In addition, they have yet to be exchanged or shared. The proposal consists of generating cryptographic keys using different configurations of artificial neural networks in order to improve the performance and reliability of the process.*

Resumo. *Com a criptografia, é possível fazer com que dados se tornem ininteligíveis para os que não tenham acesso às regras planejadas. As chaves criptográficas garantem a segurança dos dados encriptados independentemente do algoritmo utilizado. Estas chaves são, normalmente, geradas a partir de valores pseudo-aleatórios, exigindo entropia, limitando seus tamanhos por necessidade de desempenho, dentre outras consequências negativas. Além disso, as chaves ainda têm que ser trocadas ou compartilhadas. A proposta se consiste em gerar chaves criptográficas utilizando diferentes configurações de redes neurais artificiais com o intuito de melhorar o desempenho e a confiabilidade do processo.*

1. Introdução

O uso da criptografia, uma das ferramentas mais importantes da Segurança da Informação, possibilita serviços trocarem informações entre si por meio de um canal inseguro, sem que terceiros entendam as mensagens trocadas, mesmo se interceptadas [Stinson 2002].

Para que os algoritmos criptográficos possam proteger uma mensagem legível, é necessária uma chave, cujo a função é controlar a operação do algoritmo de forma que somente esta chave seja capaz de descriptografar a mensagem.

Se esta chave for descoberta ou até mesmo roubada, as consequências podem ser desastrosas, uma vez que mensagens trocadas pelo serviço que faz o uso desta chave poderão ser descriptografadas e seu conteúdo — muitas vezes privado — exposto. Logo, uma parte essencial para garantir a segurança da informação é a realização da troca das chaves de uma forma segura quando a chave utilizada é compartilhada, como por exemplo na criptografia simétrica.

Tendo em vista a importância da chave criptográfica, é notório que a principal dificuldade no desenvolvimento de sistemas seguros baseados em trocas de informações não é a escolha de um algoritmo criptográfico ideal, mas sim, ao gerenciamento da estrutura necessária para gerar, autenticar e transmitir as chaves criptográficas [Al-Riyami and Paterson 2003].

2. Revisão da teoria

2.1. Criptografia Neural

Criptografia neural é uma área da criptografia dedicada à utilizar redes neurais artificiais em procedimentos que envolvem geração e troca de chaves criptográficas. Em suas aplicações, são comumente utilizadas uma configuração específica de redes neurais artificiais chamada *Tree Parity Machine* (explicadas adiante na seção 2.2).

A técnica de geração de chaves utiliza da sincronização mutua entre duas *Tree Parity Machines* (representada pela Figura 1) inicializadas com os mesmos parâmetros. No processo de treinamento das redes neurais artificiais, o vetor de entrada ($x_{(k,j)}$) utilizado em cada iteração é compartilhado e as saídas ($\tau^{A/B}$) são comparadas para decidir se a regra de aprendizado será aplicada ou não.

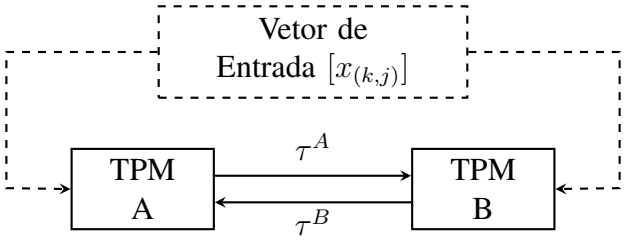

Figura 1. Sincronização entre *Tree Parity Machines*.

As redes somente atingem a sincronização quando seus vetores de pesos sinápticos (explicadas adiante na seção 2.2) se tornem idênticos. Isto é o critério de parada para o treinamento e indica que a chave foi gerada. A chave resultante é o próprio vetor de pesos sinápticos. Como ele está presente em ambas as redes, é possível instanciar cada rede em uma aplicação distinta, assim ao final do processo, ambas as redes possuirão a chave, caracterizando a troca.

2.2. Redes Neurais Artificiais

Redes neurais artificiais (RNA) são estruturas que têm como objetivo reproduzir computacionalmente o processamento realizado pelo cérebro humano. Uma RNA é uma agregação de unidades discretas de processamento baseadas em neurônios biológicos, intitulados neurônios artificiais [Aragão 2018].

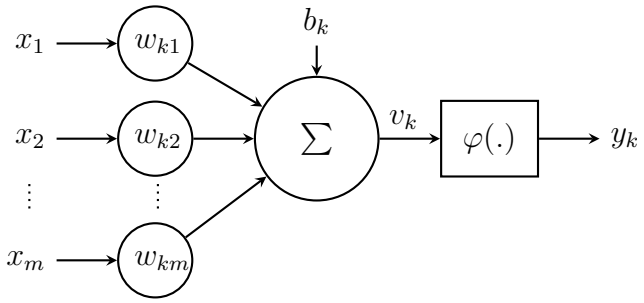

Figura 2. Representação de um neurônio artificial.

Os neurônios artificiais, representados pela Figura 2, recebem um ou mais estímulos de entrada (x_m), sendo que cada um possui uma influência sobre o processamento conhecida como "peso sináptico" (w_{km}). Estas entradas ponderadas, juntamente com um fator de correção fixo conhecido como viés ou limiar de ativação (*bias*) (b_k), que possui seu próprio peso sináptico, são combinados linearmente ($\sum$) a fim de gerar um valor denominado potencial de ativação (v_k). Este valor é utilizado como entrada da função de ativação (φ), que serve para limitar a saída (y_k) do neurônio associado.

A rede é submetida a conjuntos de entrada cujas saídas podem ser conhecidas ou não, este procedimento é conhecido como época de treinamento. Em caso afirmativo, o treinamento é dito supervisionado; caso contrário, é dito não-supervisionado. Após calcular as saídas, uma regra de aprendizado é aplicada para atualizar os pesos sinápticos até que um critério de parada seja satisfeito. A representação do conhecimento adquirido é dada, portanto, pelos valores finais dos pesos sinápticos da rede neural.

Inúmeras combinações com neurônios artificiais podem ser feitas, logo existem incontáveis configurações de redes neurais. A *Tree Parity Machine* (TPM) é um tipo característico de RNA *multilayer* (mais de uma camada de neurônios) *feedforward* (saídas sempre alimentam a próxima camada) frequentemente utilizado em aplicações que envolvem criptografia neural (vide seção 2.1).

As TPMs possuem uma única saída ($\tau \in \{-1, +1\}$) e 3 camadas (*layers*) fixas: camada de entrada (*input layer*); camada oculta ou escondida (*hidden layer*) e camada de saída (*output layer*), que contém uma função multiplicadora. Na configuração da TPM são selecionados três parâmetros:

- K: Número de neurônios na camada escondida ($1 \leq k \leq K$)
- N: Número de entradas para cada neurônio ($1 \leq j \leq N$)
- L: Configura a faixa de valores dos pesos sinápticos, na qual $w_{kj}(t) \in [-L, L]$

Em uma TPM tem-se $K \times N$ elementos gerados para compor o vetor de entrada, sendo que as entradas obedecem $x_{kj}(t) \in \{-1, +1\}$ [Volkmer and Wallner 2005]. A Figura 3 apresenta uma *Tree Parity Machine* parametrizada com $K = 3$ e $N = 4$.

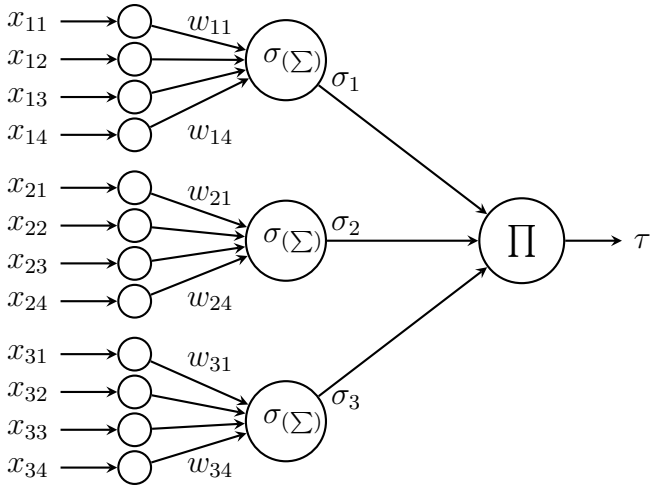

Figura 3. *Tree Parity Machine* (TPM) parametrizada com *K* = 3 e *N* = 4.

3. Revisão da bibliografia

Estudos direcionados à área de geração de chaves criptográficas por meio de criptografia neural se tornaram recorrentes em publicações a partir dos anos 2000. Os trabalhos buscaram não só melhorar o desempenho do método, como também prover mais segurança em aplicações que envolvam troca de chaves em meios de comunicação não confiáveis.

Em 2006, Ruttor [Ruttor 2006] propôs uma das mais conhecidas e utilizadas técnicas de criptografia neural: a sincronização entre duas redes neurais artificiais do tipo *Tree Parity Machines* baseando-se em seus aprendizados mútuos. Na proposta, as TPM eram inicializadas com pesos sinápticos aleatórios e compartilhavam suas saídas com o intuito de se sincronizarem. Vetores congêneres aleatórios alimentavam ambas as redes e, caso suas saídas coincidissem, a regra de aprendizado era aplicada e seus pesos eram atualizados. As RNAs estavam sincronizadas quando seus vetores de pesos sinápticos tornaram-se análogos. Tais vetores, após a sincronização, consistiram na chave criptográfica gerada pelo processo.

4. Proposta

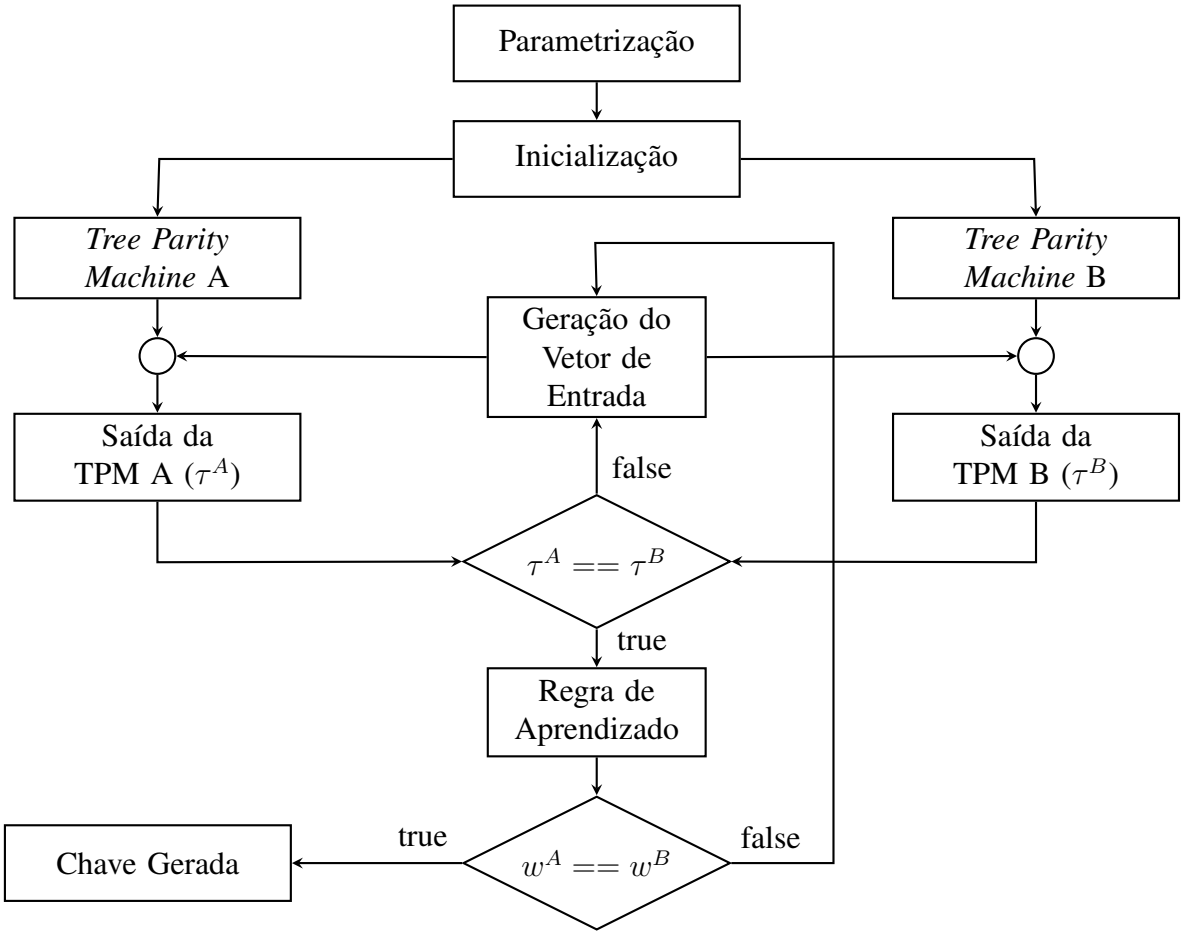

Figura 4. Diagrama em blocos da proposta.

A proposta deste trabalho, descrita pelo diagrama da Figura 4, é implementar a estrutura da rede neural artificial *Tree Parity Machine* baseada na proposta de Ruttor [Ruttor 2006]

utilizando a linguagem de programação C de tal forma que duas TPMs distintas, porém com as mesmas configurações, se convirjam com o objetivo de gerar uma chave criptográfica a partir de suas sincronizações mutuas.

Este procedimento de criptografia neural será metrificado com a intenção de otimizar o processo a partir dos parâmetros requeridos. Os blocos utilizados no diagrama da Figura 4 para modularizar a implementação estão descritos detalhadamente a seguir:

- Parametrização: Aquisição dos parâmetros necessários para instanciação das duas *Tree Parity Machines* (*K*, *N*, *L*, função de ativação e regra de aprendizado).
- Inicialização: Procedimento no qual ocorre a instanciação de duas TPMs com os mesmos parâmetros *K*, *N*, *L*, função de ativação e regra de aprendizado, porém com pesos sinápticos inicialmente distintos gerados pseudo-aleatoriamente.
- *Tree Parity Machine A* / *Tree Parity Machine B*: *Tree Parity Machines* devidamente instanciadas e inicializadas.
- Saída da TPM A (τ^A) / Saída da TPM B (τ^B): Saída das TPMs ($\tau^{A/B}$) representada pela Equação (1), onde $\sigma^{A/B}$ representa a saída dos *K* neurônios da camada escondida dada pelas suas funções de ativação, $w_{k,n}^{A/B}$ é o vetor de pesos sinápticos e $x_{k,n}^{A/B}$ representa o vetor de entrada.

$$\tau^{A/B} = \prod_{k=1}^{K} \sigma_k^{A/B} = \prod_{k=1}^{K} \sigma \left(\sum_{n=1}^{N} w_{k,n}^{A/B} \times x_{k,n}^{A/B} \right) \tag{1}$$

- Geração do vetor de entrada: Geração pseudo-aleatória de um vetor de $(K \times N) + K$ posições que será utilizado como entrada para ambas as redes neurais artificiais.
- Regra de Aprendizado: Aplicação da regra de aprendizado parametrizada para atualizar os pesos sinápticos para a próxima época de treinamento. As regras de aprendizado utilizadas estão descritas nas equações a seguir, sendo o aprendizado *Hebbian* representado pela Equação (2), *Anti-Hebbian* pela Equação (3) e *Random Walk* descrito pela Equação (4), nos quais a função $\Theta(x)$ é uma função do tipo sinal, $W_{k,n}^{A/B}$ é o novo peso sináptico atualizado com a regra de aprendizado, $w_{k,n}^{A/B}$ é o peso sináptico atual, $x_{k,n}$ é um elemento do vetor de entrada referente ao peso sináptico que será atualizado e $\tau^{A/B}$ representa a saída da rede.

$$W_{k,n}^{A/B} = w_{k,n}^{A/B} + x_{k,n}\tau^{A/B}\Theta(\tau^{A/B}\sigma_k^{A/B})\Theta(\tau^A\tau^B) \tag{2}$$

$$W_{k,n}^{A/B} = w_{k,n}^{A/B} - x_{k,n}\sigma_k^{A/B}\Theta(\tau^{A/B}\sigma_k^{A/B})\Theta(\tau^A\tau^B) \tag{3}$$

$$W_{k,n}^{A/B} = w_{k,n}^{A/B} + x_{k,n}\Theta(\tau^{A/B}\sigma_k^{A/B})\Theta(\tau^A\tau^B) \tag{4}$$

- Chave Gerada: Como os vetores de pesos sinápticos de ambas as redes se mostraram idênticos, significa que as redes convergiram. Logo a chave criptográfica gerada corresponde ao vetor de pesos sinápticos das *Tree Parity Machines* (excluindo o *bias* de cada neurônio) aplicados à função sinal ($\Theta(x)$).

5. Experimentos

5.1. Metodologia

Com a finalidade de otimizar o processo de geração de chaves criptográficas por meio da criptografia neural, experimentos foram realizados com diferentes configurações de *Tree Parity Machines* a fim de analisar as influências individuais dos parâmetros de inicialização das redes neurais artificiais utilizadas no procedimento.

Os parâmetros analisados foram:

1. Variação de *K*: Número de neurônios na camada escondida.
2. Variação de *N*: Número de entradas para cada neurônio da camada escondida.
3. Variação de *L*: Faixa de valores discretos possíveis para os pesos sinápticos.
4. Regra de Aprendizado:
   (a) *Hebbian**
   (b) *Anti-Hebbian**
   (c) *Random-Walk**

   * Descrito detalhadamente na seção 4.
5. Função de Ativação: As funções de ativação utilizadas estão descritas nas equações a seguir, sendo elas:
   (a) Sinal:

$$\varphi(x) := \begin{cases} -1 & \text{if } x \leq 0 \\ +1 & \text{if } x > 0 \end{cases} \tag{5}$$

   (b) Sigmóide:

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

   (c) Linear:

$$\varphi(x) = x \tag{7}$$

   (d) Tangente Hiperbólica (tanh):

$$\varphi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{8}$$

Para garantir consistência nos resultados obtidos e compatibilidade entre todos os experimentos realizados, o gerador de números pseudo-aleatórios teve sua semente de geração fixada em *962611861*. Além disso, cada configuração de *Tree Parity Machine* utilizada nos experimentos gerou 100 chaves binárias consecutivas de tamanho notavelmente superior quando comparadas às chaves utilizadas nos algoritmos criptográficos comumente utilizados com o objetivo de salientar os resultados e tornar suas visualizações mais inteligíveis.

As métricas finais coletadas nas gerações sucessivas de chaves foram obtidas por meio de uma média aritmética dos valores obtidos no processo de geração de cada uma das 100 chaves.

As métricas designadas para análise foram:

1. Chaves Repetidas: Número de chaves repetidas no conjunto gerado juntamente com a probabilidade de ocorrência de cada chave (P_k), descrita pela Equação (9).

$$P_k[\%] = \frac{1}{2^{(K \times N)}} \times 100 \tag{9}$$

2. Épocas de Treinamento: Números médio, máximo e mínimo de épocas de treinamento necessárias para que ocorra a sincronização entre as duas *Tree Parity Machines* envolvidas no processo de geração das chaves.
3. Volume de Dados: O volume de dados (V) trocados entre as duas RNAs (em *Bytes*) em função da média de épocas de treinamento (*epochs_avg*) e dos parâmetros da TPM, descrito pela Equação (10). Bastante relevante para analisar situações onde as duas *Tree Parity Machines* não se encontram no mesmo sistema, ou até mesmo na mesma aplicação.

$$V[Bytes] = \frac{epochs\_avg \times [(K \times N) + K + 2]}{8} \tag{10}$$

4. Comprimento do vetor de entrada: O número de elementos no vetor de entrada ($I_{[K,N}$) é o resultado da soma de todas as entradas de cada neurônio da camada escondida ($K \times N$) com o acréscimo de uma entrada para o *bias* de cada um dos neurônios (K).

$$I_{[K,N]} = (K \times N) + K \tag{11}$$

5. Tempo de Geração: Tempos médio, máximo e mínimo tal como o desvio padrão do processo de geração de cada uma das 100 chaves separadamente, desde a inicialização das *Tree Parity Machines* até o alcance da sincronização.

Para aferir a influência das 3 diferentes regras de aprendizado e das 4 diferentes funções de ativação, os parâmetros *K*, *N* e *L* foram fixados em, respectivamente, 32, 32 e 5. Logo cada uma das 100 chaves criptográficas binárias geradas possuem um tamanho imutável de 1Kib (1Kib $\equiv$ 1.024 *bits*).

A rotina de ensaio das 12 possíveis combinações entre regras de aprendizado e funções de ativação que foram utilizados no experimento foi intitulada "Experimento 1" e é representada na Figura 5.

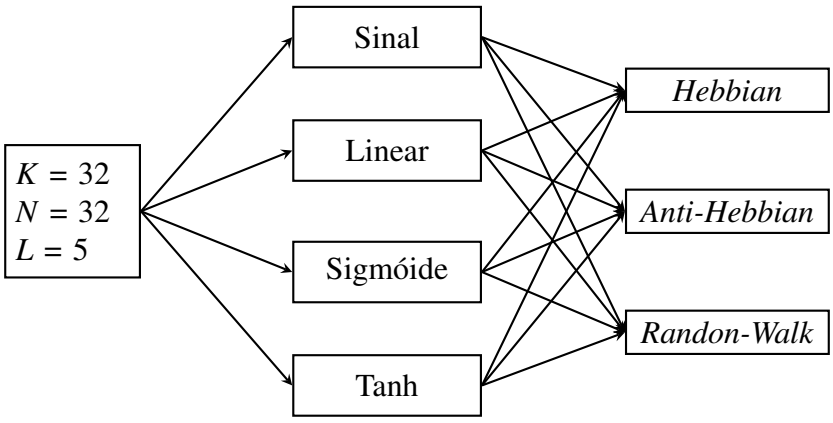

Figura 5. Experimento 1.

Após a realização do Experimento 1, foram analisados os resultados e foram identificados, para esta implementação, os 3 conjuntos de regra de aprendizado e função de ativação que obtiveram maior desempenho relativo na geração de chaves (resultado discutido detalhadamente na seção 5.2.1). Logo, nas rotinas implementadas para análise dos

parâmetros *K*, *N* e *L*, as combinações tidas como mais eficientes foram utilizadas nas estruturas de análise dos mesmos.

Para as análises de *K* e *N*, o experimento realizado foi denominado “Experimento 2” e está representado na Figura 6. Nesta rotina, *L* foi fixado em 5 e um conjunto finito de pares de valores inteiros ($C_i[k_i, n_i]$) onde seus produtos resultam em 1Kib foi atribuído aos valores dos dois parâmetros.

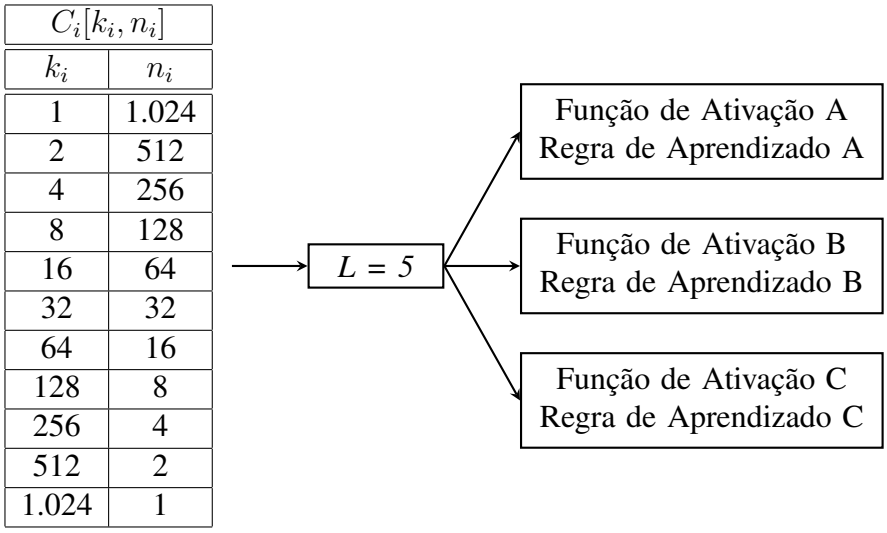

Figura 6. Experimento 2.

O objetivo de rodar a criptografia neural utilizando o conjunto *C* é analisar a sensibilidade da implementação ao gerar chaves com exatamente o mesmo tamanho (1Kib) porém em configurações diversas. Com isso, espera-se concluir a relação entre número de entradas e número de neurônios artificiais para gerar chaves equivalentes entre si.

Já na análise de *L*, o parâmetro foi variado entre 2 e 50 enquanto os demais foram fixados em seus valores ótimos. Esta sequencia de execuções, representada na Figura 7, foi denominada “Experimento 3”.

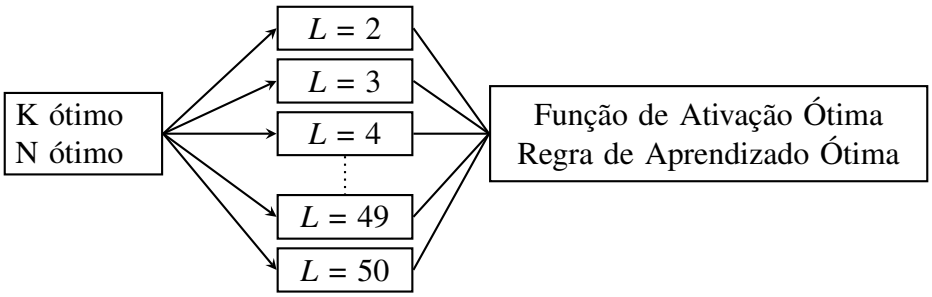

Figura 7. Experimento 3.

Com os parâmetros individualmente analisados, é possível otimizar os parâmetros de geração das chaves a partir do tamanho de chave requerido.

Para gerar os parâmetros necessários na criptografia neural a partir do tamanho da chave desejada, foram utilizados os resultados dos 4 experimentos realizados que foram descritos detalhadamente nesta seção.

Para asseverar a eficiência do método, foram gerados dois grupos de chaves criptográficas binárias de 10Kib (tamanho 10 vezes maior do que o tamanho utilizado nos demais experimentos). Um grupo com 1.000 (mil) chaves e outro com 10.000 (dez mil) chaves. Este procedimento foi alcunhado "Experimento 4" e é discutido na seção 5.2.4.

5.2. Resultados

Com o objetivo de facilitar o entendimento e a compreensão dos resultados dos experimentos executados, as discussões foram divididas como sendo uma subseção referente a cada um dos 4 experimentos realizados conforme a seguinte distribuição:

- Experimento 1 (Seção 5.2.1)
- Experimento 2 (Seção 5.2.2)
- Experimento 3 (Seção 5.2.3)
- Experimento 4 (Seção 5.2.4)

5.2.1. Experimento 1

Na análise dos resultados obtidos ao fim do Experimento 1 (Figura 5), 5 das 12 configurações possíveis entre regras de aprendizado e funções de ativação foram descartadas das avaliações. Dois motivos foram responsáveis pela rejeição, o primeiro foi o fato das *Tree Parity Machines* em duas das cinco configurações descartadas divergirem, logo não atingindo o sincronismo e consequentemente não gerando as chaves. O segundo foi que, em três configurações, embora as redes alcançassem o sincronismo e gerassem as chaves, todas as 100 chaves geradas eram iguais com todos os 1.024 *bits* em 1 ou em 0, assim inviabilizando sua utilização. As configurações descartadas juntamente com o motivo de suas rejeições estão descritos na Tabela 1.

Tabela 1. Configurações descartadas de análises pelo Experimento 1.

Função de Ativação	Regra de Aprendizado	Motivo do Descarte
Sigmóide	*Random Walk*	Não convergiu
Tangente Hiperbólica	*Random Walk*	Não convergiu
Sigmóide	*Hebbian*	Chaves repetidas
Linear	*Hebbian*	Chaves repetidas
Tangente Hiperbólica	*Hebbian*	Chaves repetidas

Levando em consideração as 7 configurações válidas, não houveram ocorrências de chaves repetidas em nenhum dos conjuntos de 100 chaves criptográficas binárias geradas por cada uma destas 7 configurações. A partir dos dados gerados nos conjuntos em questão foram analisadas as métricas propostas na seção 5.1.

Tabela 2. Tabela de resultados do Experimento 1.

Resultados do Experimento 1						
Função de Ativação	Regra de Aprendizado	Número de Épocas		Tempo [microsec]		Volume de Dados [*Bytes*]
		médio	σ	médio	σ	
Sinal	*Hebbian*	79	16,6	6.352	1.404,7	10.448
	Anti-Hebbian	21	5	1.765	423,9	2.777
	Random Walk	86	22,2	6.787	1.745,2	11.373
Linear	*Anti-Hebbian*	8	1,9	876	501,6	1.058
	Random Walk	55	14	5.845	1.600,6	7.274
Sigmóide	*Anti-Hebbian*	16	3,9	1.748	551,9	2.116
Tanh	*Anti-Hebbian*	67	21,5	7.503	2.434,7	8.860

A abordagem utilizada para definir a combinação ótima entre regra de aprendizado e função de ativação foi atingir a maior eficiência possível no processo de geração. Não foram levados em consideração fatores como precauções de segurança como, por exemplo, prevenção aos ataques tratados na seção 6.1.

Analisando os resultados obtidos neste experimento, pôde-se afirmar que, para esta aplicação, as 3 combinações de função de ativação e regra de aprendizado que se mostraram mais eficiente foram: [sinal, *Anti-Hebbian*], [*linear*, *Anti-Hebbian*] e [sigmóide, *Anti-Hebbian*]. Isto porque estes conjuntos demandaram menos épocas de treinamento para sincronizar, fazendo com que menos dados fossem trocados entre as duas *Tree Parity Machines* e, consequentemente, que seus tempos de execução fossem menores do que as demais configurações, atingindo assim uma maior eficiência no processo.

Uma vez que todas as três combinações que obtiveram maior desempenho no processo de geração das chaves utilizam a regra de aprendizado *Anti-Hebbian*, pode-se concluir que esta é a regra considerada ótima para esta implementação.

5.2.2. Experimento 2

Após a execução, duas das três combinações de funções de ativação com regra de aprendizado foram descartadas das próximas análises. Os dois motivos responsáveis por esta ação foram os mesmos encontrados no Experimento 1, sendo eles, a divergência entre as redes neurais artificiais em determinadas configurações de K e N ou a geração das 100 chaves idênticas com os 1.024 *bits* em 1 ou em 0. As configurações descartadas juntamente com o motivo de suas rejeições estão descritas na Tabela 3.

Tabela 3. Configurações descartadas de análises pelo Experimento 2.

Função de Ativação	Regra de Aprendizado	Motivo do Descarte
Sigmóide	*Anti-Hebbian*	Não convergiu
Linear	*Anti-Hebbian*	Chaves repetidas

A partir desta análise, foi constatado que o conjunto [sinal, *Anti-Hebbian*] se mostrou ótimo para esta implementação. Isto se deu ao fato do conjunto ser o mais rápido no que se refere ao tempo total de geração por chave binária, e ao mesmo tempo ser resiliente às 11 diferentes combinações de *K* e *N* presentes no conjunto *C* que resultam em chaves de 1Kib propostas pelo Experimento 2.

Para obter a proporção ótima entre os parâmetros *K* e *N*, a rotina do Experimento 2 foi aplicada utilizando a regra de aprendizado *Anti-Hebbian* e a função de ativação sinal. Os valores das métricas estão dispostos na Tabela 4.

Tabela 4. Tabela de resultados do Experimento 2.

Resultados do Experimento 2						
$C_i[k_i, n_i]$		Épocas		Tempo [micro]		Volume de Dados [B]
k_i	n_i	médio	σ	médio	σ	
1	1.024	15	3,8	1.188	308,8	1.925
2	512	16	3,1	1.210	209,9	2.056
4	256	18	3,9	1.263	305,2	2.317
8	128	18	3,6	1.325	306,8	2.455
16	64	19	4,5	1.570	427,9	2.605
32	32	21	5	1.453	309	2.645
64	16	21	5,2	1.575	350,4	2.861
128	8	23	5,4	1.751	411	3.317
256	4	22	5,9	2.010	413	3.685
512	2	23	5,4	2.485	506,7	4.421
1.024	1	15	3	2.431	440,9	3.843

Levando em consideração o número de épocas para sincronizar, ambos os pares [1, 1.024] e [1.024, 1] obtiveram o mesmo desempenho. Entretanto, pode-se concluir que os valores de *K* e *N* que se mostraram mais eficientes foram $K = 1$ e $N = 1.024$.

Quando o tempo de geração individual das chaves e o volume de dados trocados entre as duas redes neurais artificiais são analisados, é possível constatar que quanto maior o número de neurônios na camada escondida, maior o tempo de geração e também do volume de dados trocados.

Este comportamento é explicado pela adição do *bias* em cada neurônio da camada escondida. Embora as chaves geradas possuam os mesmos tamanhos (1Kib), os vetores de entrada que são utilizados no treinamento das *Tree Parity Machines* têm seus tamanhos variados dependendo do número de neurônios na camada escondida uma vez que é necessário gerar uma entrada para cada *bias* além do número de entradas. A comparação dos cálculos é demonstrada pelas Equações (12) e (13).

$$I_{[1,1.024]} = (1.024 \times 1) + 1 = 1.025 \tag{12}$$

$$I_{[1.024,1]} = (1 \times 1.024) + 1.024 = 2.048 \tag{13}$$

A mesma conduta é identificada no cálculo do volume de dados. Esta está representada nas Equações (14) e (15).

$$V_{[1,1.024]} = \frac{15 \times [(1 \times 1.024) + 1 + 2]}{8} = 1.925[B] \tag{14}$$

$$V_{[1.024,1]} = \frac{15 \times [(1.024 \times 1) + 1.024 + 2]}{8} = 3.843[B] \tag{15}$$

Neste ponto pode-se concluir que uma *Tree Parity Machine* com um neurônio na camada escondida se torna mais eficiente no que se refere ao tempo de geração de chaves. Por conseguinte, o parâmetro *K* foi fixado em 1.

Como *K* foi fixado em 1 e o tamanho da chave gerada pela criptografia neural é obtido por meio do produto entre o número de neurônios na camada escondida com o número de entradas de cada neurônio (excluindo o *bias*). Tem-se que o valor de *N* será correspondente ao tamanho da chave a ser gerada.

5.2.3. Experimento 3

Na execução das iterações propostas no experimento, os valores de *L* foram variados de 2 à 50, enquanto os demais parâmetros foram fixados nos resultados ótimos aferidos pelos experimentos anteriores. São estes parâmetros: $K = 1$, $N = 1.024$, função de ativação sinal e regra de aprendizado *Anti-Hebbian*.

Após a realização do ensaio, a influência do parâmetro *L* no processo de criptografia neural nas condições citadas anteriormente foi representado de forma discreta no gráfico da Figura 8.

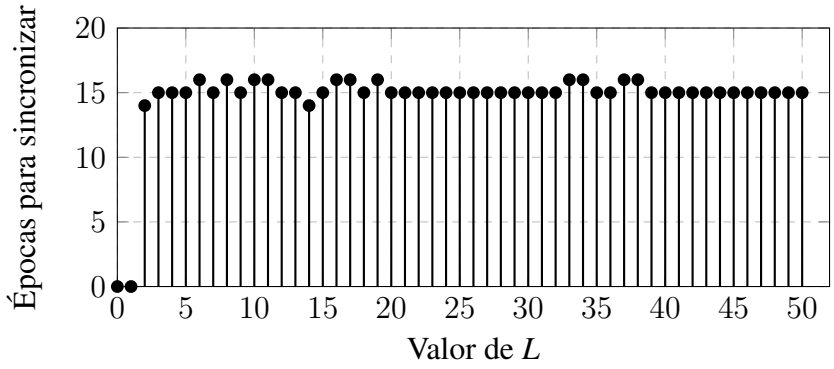

Figura 8. Análise da influência de *L* nas condições alcançadas.

Com os resultados do Experimento 3 é possível concluir que o parâmetro *L* não interfere no desempenho do processo de geração de chaves criptográficas binárias para esta configuração de *Tree Parity Machine* alcançada. Tendo em vista esta situação, o parâmetro *L* foi fixado em 2, que é o menor valor tangível para o mesmo.

5.2.4. Experimento 4

Baseando-se nos experimentos anteriores, a estrutura que foi constituída ótima possui a seguinte parametrização:

- K: 1
- N: Tamanho da chave desejada
- L: 2
- Função de Ativação: Sinal
- Regra de Aprendizado: *Anti-Hebbian*

Utilizando esta configuração ótima, os dois grupos de chaves propostos pelo Experimento 4 foram geradas, tal qual a coleta das métricas estudadas. Em seguida seus resultados foram gerados e estão representados pela Tabela 5 e pelo gráfico da Figura 9.

Tabela 5. Tabela de resultados do Experimento 4.

Resultados do Experimento 4						
Configuração Ótima	Número de Chaves	Número de Épocas		Tempo [microsec]		Volume de Dados [B]
		médio	σ	médio	σ	
K = 1 N = 10.240 L = 2 Sinal *Anti-Hebbian*	1.000	18	3	14.080	1.928,6	23.046
	10.000	18	3	14.016	1.865	23.046

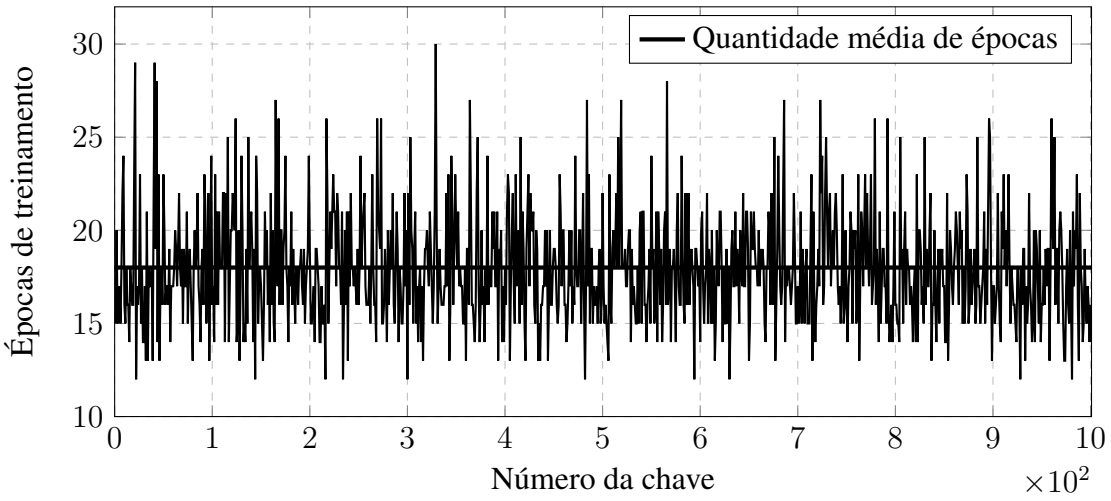

Figura 9. Épocas de treinamento na geração das 1.000 chaves binárias.

6. Conclusão

Baseando-se na proposta de Ruttor [Ruttor 2006] que descreveu a troca e geração de chaves criptográficas binárias por meio de redes neurais artificiais do tipo *Tree Parity Machine*, foi realizada uma implementação do mecanismo na linguagem C com o intuito

de aferir o seu real desempenho. A partir desta implementação, 4 experimentos foram elaborados e realizados para que uma configuração de *Tree Parity Machine* ideal seja alcançada.

Como resultado geral, uma configuração ótima foi adotada. O aferimento da validade desta configuração foi que na geração sequencial de dois grupos de chaves criptográficas binárias de 10Kib (tamanho hiperbólico quando comparado ao tamanho das chaves comumente utilizadas), obteve-se um tempo médio de geração por chave de 14.080 microssegundos no grupo de mil chaves e de 14.016 microssegundos para o grupo de dez mil chaves. Além disso não houve ocorrência de chaves repetidas em nenhum dos dois conjuntos.

6.1. Trabalhos Futuros

No decorrer do estudo, foram identificados pontos que condescendem possíveis extensões na pesquisa. Um dos principais pontos a serem estudados seria a vulnerabilidade da estrutura alcançada aos ataques clássicos à este tipo de implementação. Em seguida, com os resultados obtidos, realizar uma criptoanálise e identificar falhas de segurança e, consequentemente, realizar possíveis melhorias na implementação atual para torná-la viável em ambientes inseguros.

Outro ponto relevante a ser estudado seria implementar e analisar o desempenho de um mecanismo de realimentação (*feedback mechanism*) nas redes neurais artificiais, como proposto por Ruttor [Ruttor 2006]. Isto acarretaria em uma entrada secreta para cada neurônio na camada escondida que não seria transmitido entre as *Tree Parity Machines*, aumentando a segurança do método proposto, diminuindo o tamanho dos vetores de entrada necessários e, consequentemente, diminuindo o volume de dados trocados entre as redes neurais artificiais.

Referências

Al-Riyami, S. S. and Paterson, K. G. (2003). Certificateless Public Key Cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 452–473. Springer.

Aragão, M. V. C. (2018). Estudo e avaliação de classificadores para um sistema anti-spam. *UNIFEI*.

Ruttor, A. (2006). Neural synchronization and cryptography. *arXiv preprint arXiv:0711.2411*.

Stinson, D. R. (2002). *Cryptography: Theory and Practice, Third Edition*. Chapman and Hall/CRC.

Volkmer, M. and Wallner, S. (2005). Tree parity machine rekeying architectures. *IEEE Transactions on Computers*, 54(4):421–427.