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

Gustavo P. de O. Celani[1]

[1] Instituto Nacional de Telecomunicações (INATEL)
Santa Rita do Sapucaí - MG, Brasil

gustavo_celani@hotmail.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

Após a revolução industrial os produtos passaram a apresentar grande semelhança por conta da maior otimização oferecida por Taylor na produção de itens idênticos. Com isto, clientes passaram a buscar diferenciais nos produtos que consumiam e, portanto, a empresa oferecesse diferenciais em suas mercadorias acabaria se destacando no mercado. Para suprir tal necessidade, as instituições passaram a buscar cada vez mais contribuidores com conhecimentos nas mais variadas áreas para que novas ideias surgissem, resultando em produtos inovadores [Antunes and Martins 2002].

A soma do conhecimento individual dos funcionários, suas experiências e *know-how*, dados gerados por serviços, informações, procedimentos, rotinas e até mesmo documentações da instituição, formam o conhecido patrimônio intelectual, atualmente considerado o mais valioso no contexto empresarial.

Este foi o divisor de águas da transição da sociedade entre a Era Industrial e a Era da Informação, na qual o conhecimento acumulado é tido como uma riqueza maior do que bens materiais. Neste novo contexto, a maior parte dos esforços passaram a ser direcionados ao tratamento, armazenamento e preservação destes dados de forma segura.

Para a maioria das instituições ativas, perder seus dados seria algo catastrófico, levando a empresa a perder registros de funcionários, clientes, fornecedores, contabilidade, dentre outras inúmeras avarias. Seu prejuízo pode ser tão alto que a única saída seria declarar falência. Como foi o caso de uma *exchange* de bitcoin sul-coreana que declarou falência após perder 17% de suas reservas de ativos [Tolotti Umpieres 2017].

Em 2014, a EY Brasil divulgou uma pesquisa sobre segurança de dados nas empresas brasileiras. Os dados mostravam que 54,2% dos empresários afirmaram que os riscos de ataques cibernéticos aumentaram no ano de 2013. Ao mesmo tempo, mais de 62% ampliariam seus investimentos em segurança da informação [Redação 2014]. Entretanto, tais esforços aparentemente não vêm sendo suficientes. De acordo com a Gemalto, mais de 2,6 bilhões de dados foram roubados, perdidos ou expostos mundialmente em 2017, constituindo um aumento de 88% em relação a 2016 [Redação 2017b].

Os usuários finais também são alvos dos crimes cibernéticos, direta ou indiretamente. De forma indireta, dados pessoais — como fotos, arquivos salvos em nuvem, *e-mails*, senhas, números de cartões, dentre outras informações — podem ser vazados da base de dados de um sistema cujo o usuário possui um cadastro. Em 2016, a empresa Uber foi alvo de um ataque *hacker black hat*[1] (também conhecidos como *full-fledged* ou *crackers* [Nakamura and Geus 2007]) que expôs dados pessoais de 57 milhões de usuários cadastros, tanto de clientes quanto de motoristas [Redação 2017a].

Com o objetivo de proteger e preservar os dados de malfeitores, a área de Segurança da Informação (SI) se tornou indispensável na Era da Informação e é atualmente normatizada pela ISO[2]/IEC[3] 27002:2013 [ISO/IEC 2013].

A criptografia é uma das ferramentas mais importantes da SI, uma vez que a técnica apoia-se nos seis atributos básicos ilustrados na Figura 1. A confidencialidade limita o acesso a informação tão somente para aqueles que possuem a devida autorização do proprietário da informação; a integridade garante que a informação permaneça isenta de qualquer alteração; a disponibilidade certifica a acessibilidade da informação para quem seja autorizado à utilizá-la; a autenticidade assegura a fonte da informação; a irretratabilidade impossibilita que a autoria da transação seja negada; a conformidade garante que as convenções estabelecidas serão seguidas de forma correta.

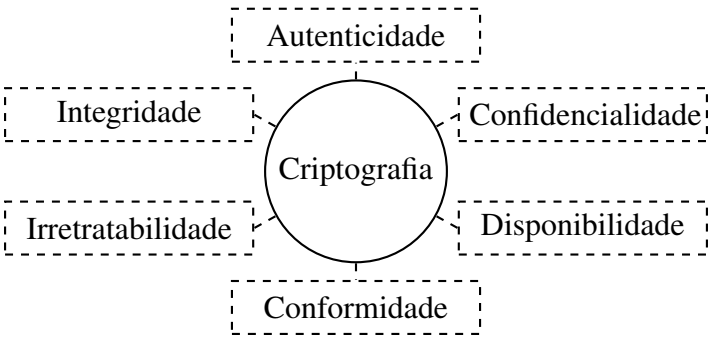

Figura 1. Criptografia apoiada nos pilares da SI.

[1] *Hackers* que utilizam das vulnerabilidades de tal forma a suplantar os limites da lei.
[2] Do inglês, *International Organization for Standardization.*
[3] Do inglês, *International Electrotechnical Commission.*

O uso da criptografia possibilita serviços a trocarem informações entre si por meio de um canal inseguro, muitas vezes a Internet, 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. A Figura 2 apresenta um cenário típico de uma criptografia simétrica, onde a chave é compartilhada.

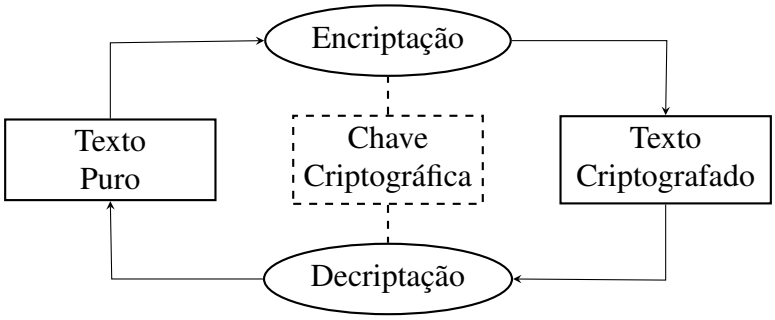

Figura 2. Compartilhamento de chave em criptografia simétrica.

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.

Isto se evidencia no caso onde o certificado digital do banco Inter foi revogado após o mesmo ter sido publicado, juntamente com sua chave privada. A revogação foi caracterizada por *key compromise*, que designa o vazamento da chave [Redação 2018]. Em um caso como este, a chave poderia ter sido usada para obtenção dos dados pessoais dos clientes do banco virtual, ou até mesmo para efetuar transações monetárias indiscriminadamente.

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].

A maneira mais comum de geração destas chaves baseia-se na pseudoaleatoriedade, de forma que nenhuma estrutura matemática ou lógica seja utilizada para recriá-la. Para que esta geração seja o mais aleatória possível, a RFC4086[4] é responsável por padronizar esta geração pseudo-aleatória [Eastlake et al. 2005].

Para garantir a pseudoaleatoriedade, é necessário coletar entropia[5] do sistema

[4]Do inglês, *Request for Comments*: documentos técnicos desenvolvidos e mantidos pelo grupo IETF (do inglês, *Internet Engineering Task Force*, responsável, por exmeplo, pela padronização de protocolos para a Internet.

[5]Informações coletadas do sistema ditas probabilisticamente aleatórias.

[Ristenpart and Yilek 2010]. Normalmente, ela é capturada por meio de eventos tratados como aleatórios como, por exemplo, as coordenadas de cliques do *mouse*, movimentos do dispositivo de disco, consulta a uma posição de memória qualquer, dentre outros eventos imprevisíveis [Young and Yung 2004]. Esta situação faz com que, para gerar uma chave, seja necessária um quantidade de entropia do sistema diretamente proporcional ao tamanho da chave. Além disso, em sistemas onde não há muitos periféricos conectados ou fontes de entropia de um modo geral, esta geração acontece de forma ineficiente.

Baseando-se no conteúdo até então apresentado, a proposta deste trabalho consiste na geração de chaves criptográficas binárias de tamanhos arbitrários utilizando redes neurais artificiais, bem como na análise de cada parâmetro utilizado no processo a fim de otimizá-lo.

1.1. Objetivos

Este trabalho tem como objetivo gerar chaves criptográficas binárias de tamanho arbitrário utilizando criptografia neural. Este procedimento será metrificado com base nos parâmetros necessários para sua realização com a intenção de otimizar o método a partir do tamanho configurado da chave.

1.2. Contribuições do trabalho

Os estudos deste trabalho visam apontar potenciais melhorias nos modelos atualmente propostos de geração de chaves criptográficas binárias.

2. Revisão da teoria

2.1. Segurança da Informação

Informação é tudo aquilo considerado relevante o suficiente para ser processado ou armazenado, tanto por humanos, quanto por máquinas. Uma mesma informação possui relevância diferente dependendo de quem a interpreta ou de como é representada.

Exemplo: um conjuntos de dados (informações) referente a uma lista de contatos não teria significado para um humano se fosse representado em forma binária, mas teria caso a representação se desse por meio de caracteres alfanuméricos. Em contrapartida, para que esta mesma lista de contatos seja interpretada por um sistema digital, ela deve estar representada em linguagem de máquina para que estes dados possam ser utilizados em algum serviço. Estas diferenças de representação são ilustradas na Figura 3.

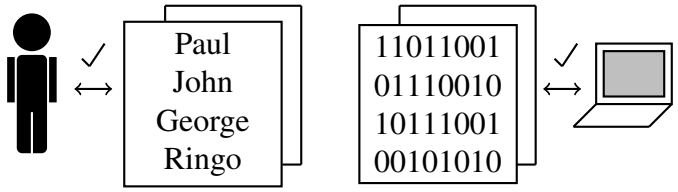

Figura 3. Diferentes interpretações da informação.

Informação pode ser algo público, como a lista de contatos exemplificada anteriormente, ou dados privados, tais como dados bancários, históricos de transações e localizações, endereços Web visitados, pesquisas realizadas, produtos comprados, número

de documentos, relação de bens, endereços, fotos, vídeos, dentre outros. Tendo em vista que o volume de dados vêm crescendo a taxas cada vez maiores [Adshead, Antony 2017], surge a necessidade de manter estes dados protegidos contra o uso ou acesso não autorizado; é neste contexto que faz-se relevante o conceito de segurança da informação (SI).

A segurança da informação é padronizada pela norma ISO/IEC 27002:2013 [ISO/IEC 2013]. Nela são definidas as propriedades básicas da SI, sendo elas:

- Confidenciabilidade: resguardo e proteção da informação apenas para quem é autorizado a acessá-la.
- Integridade: garantia da consistência física e/ou lógica dos dados durante toda sua existência.
- Disponibilidade: manutenção da disponibilidade das informações para o maior número de serviços/pessoas possíveis durante o máximo de tempo concebível.
- Autenticidade: garantia que o dado a ser interpretado é íntegro.
- Irretratabilidade ou Não Repúdio: garantia de que o gerador da informação não é capaz de negar sua autoria.
- Conformidade: certeza de que as partes envolvidas seguirão as convenções e regulamentos estabelecidos.

Caso alguma propriedade básica da SI seja desconsiderada ou transgredida, a informação resguardada deixará de ser considerada segura, o que pode acarretar em consequências financeiras, éticas ou morais. Uma série de recursos podem ser utilizados com a finalidade de evitar a violação destas propriedades. Existem mecanismos físicos, como infraestrutura, blindagem, restrição de acesso, dentre outros. Há também os controles lógicos que, dentro de uma gama de métodos existentes, incluem protocolos de comunicação, mecanismos de certificação digital, garantia de integridade por meio de técnicas de *hashing*[6], *firewall*[7] e *proxy*[8] de rede, anti-vírus, assinatura digital e criptografia [Saint-Germain 2005].

2.2. Criptografia

A criptografia é a ciência da escrita secreta [Rob 1982]. A necessidade do homem de poder se comunicar sem que as mensagens trocadas fossem interpretadas é antiga. Pode-se observar essa imprescindibilidade da criptografia em cenários como guerras, onde havia necessidade de comunicação com os combatentes sem que os opositores entendessem o que foi passado a eles, ou em situações nas quais é preciso limitar a informação somente para quem é autorizado a obtê-la.

Um dos métodos criptográficos mais tradicionais é o da cifragem simples, que se tornou icônico quando utilizado pelo ex-ditador romano Júlio César para se comunicar com seus subordinados. Neste método de substituição monoalfabético[9], os caracteres das mensagens são substituídos por outros seguindo um número de deslocamento padrão pré-determinado [Cohen 1995].

[6]Algoritmos que mapeiam dados de diversos comprimentos para dados de comprimento fixo.

[7]Dispositivo de rede que visa aplicar uma política de segurança na mesma.

[8]Agente intermediário em uma rede que age nas requisições dos clientes.

[9]Cifra de substituição onde cada letra do texto em claro é substituída por uma outra letra no texto cifrado, de forma constante.

O método é exemplificado na Figura 4 utilizando a chave “2”, ou seja, com deslocamento de duas posições. Foi considerado um alfabeto de 26 letras. $E_n(x)$ representa a função de encriptação e $D_n(x)$ representa a função de decriptação, onde x é o caractere alfabético.

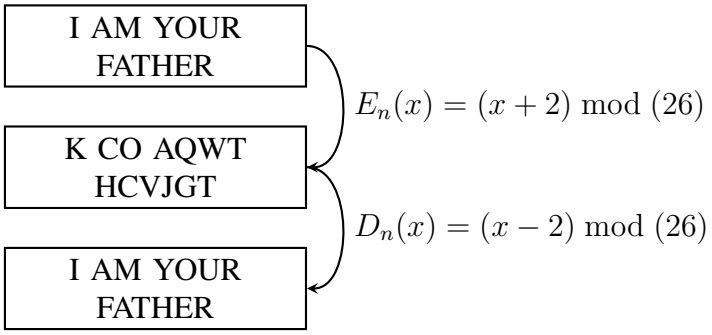

Figura 4. Cifra de César utilizando a chave "2".

Com o recorrente uso de métodos criptográficos nos mais diversos ecossistemas, a criptografia foi adquirindo importância e atraindo o interesse de acadêmicos e entusiastas, implicando no surgimento de novas técnicas cada vez mais aprimoradas.

Como vários pesquisadores passaram a contribuir com o desenvolvimento da criptografia, termos específicos passaram a ser utilizados com fins de padronização. Dentre eles, destacam-se [Rob 1982][Stallings 2011]:

- Texto puro (*plain text*): Nome dado à mensagem original, sem nenhum tipo de codificação.
- Texto criptografado (*encrypted text*): Se refere à mensagem codificada.
- Encriptação (*encryption*): Ato de codificar o texto puro para o texto criptografado.
- Decriptação (*decryption*): Transformação do texto criptografado de volta ao texto puro.
- Chave criptográfica (*cryptographic key*): Parâmetro utilizado pelos algoritmos de encriptação/decriptação para realizarem suas respectivas funções (explicadas com mais detalhes na seção 2.3).

Após o esforço realizado para melhorar cada vez mais as técnicas criptográficas devido a eminente demanda pela segurança da informação, algoritmos relevantes passaram a surgir. Dentre esses algoritmos, podem ser citados:

2.2.1. *Data Encryption Standard* (DES)

Seu contexto é dado pela necessidade que o *National Bureau of Standards* (NBS), atual *National Institute of Standards and Technology* (NIST), instituto norte-americano padronizador de tecnologias, sentiu de proteger seus dados virtuais. Para tal, foi criado um algoritmo padrão capaz de proteger dados tanto nas transmissões quanto no armazenamento. A empresa norte-americana IBM (International Business Machines) propôs o Lucifer, um algoritmo de cifragem em blocos de 128 *bits* utilizando uma chave, também, de 128 *bits* [Terada 2008]. O algoritmo de criptografia simétrica passou por uma série

de modificações de terceiros, incluindo diminuição da chave para 64 *bits* (sendo 56 *bits* efetivos e 8 *bits* de paridade com objetivo de controle), e fora publicado em detalhes pela *National Security Agency* (NSA) em 1975. Entretanto, somente no ano seguinte ele foi aceito como padrão e passou a ser chamado de DES [Grabbe 1992].

Na encriptação utilizando DES (Figura 5) o texto puro é submetido a uma permutação inicial (*Initial Permutation - IP*), o resultado da *IP* passa por uma sequência de 16 estágios da Função Feistel (*Feistel-F*). A chave criptográfica é então escalonada gerarando 16 sub-chaves de 48 *bits*, cada uma usada em um estágio. Com o resultado de cada *F* é realizada uma operação lógica OU-EXCLUSIVO (*XOR*) com o anterior. Por fim, o fruto gerado pela sequência de *XOR* é submetido a uma permutação final (*Final Permutation - FP*), produzindo assim assim o texto criptografado. Para decriptar a mensagem, basta utilizar a mesma estrutura, porém com as sub-chaves geradas espelhadas, ou seja, a *F1* receberá a sub-chave 16 e assim sucessivamente [Masucci 2003].

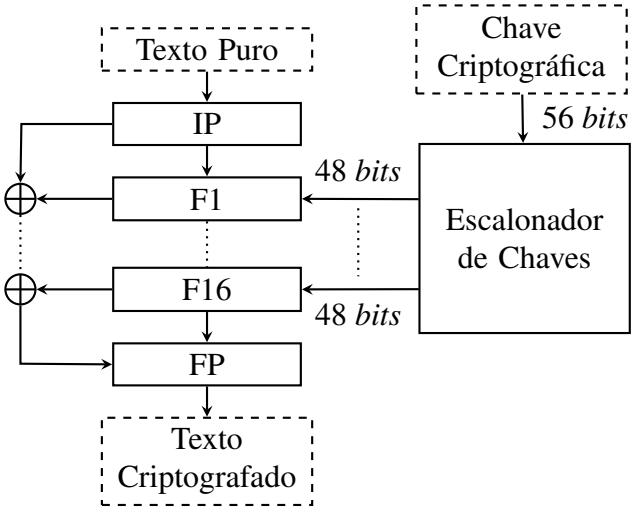

Figura 5. Procedimento de encriptação utilizando DES.

2.2.2. *Blowfish* e *Twofish*

O funcionamento do algoritmo é semelhante ao DES, porém, comporta chaves de 32 *bits* até 448 *bits*, tornando-o maleável para ser aplicado em diferentes finalidades [Thakur and Kumar 2011].

Já o algoritmo *Twofish* foi uma proposta baseada no *Blowfish*, porém, utiliza chaves de até 256 *bits*, onde metade é utilizada como chave simétrica e a outra metade é usada para remodelar o algoritmo em cada iteração das funções Feistel [Lucks 2001].

2.2.3. 3DES

O 3DES, ou *Triple DES*, consistem em três iterações do algoritmo DES, onde a saída de um serve de entrada para o outro. O texto puro é entrada para o primeiro DES, cuja saída é entrada para o segundo DES; analogamente, o terceiro DES utiliza a saída do segundo como entrada e, finalmente, a saída do terceiro DES será o texto criptografado.

2.2.4. Cast-128 e Cast-256

Baseado no algoritmo DES, o algoritmo *Cast-128* – também conhecido como *Cast5* – foi publicado em 1997. Este também faz o uso dos 16 *rounds* das funções Feistel, porém, opera com blocos de 64 *bits* e suporta chaves de 40, 64, 80 ou 128 *bits* [Adams 1997].

No ano de 1999, foi formalizado o algoritmo *Cast-256* (ou *Cast6*). Este assume blocos de 128 *bits* com chaves de 128, 160, 192, 224, ou 256 *bits* [Adams and Gilchrist 1999]. Ele realiza 48 *rounds*, ou 12 "*quad-rounds*" da função Feistel, onde as sub-chaves escalonadas são de 32 *bits* [Riaz and Heys 1999].

2.2.5. *Advanced Encryption Standard* (AES)

O algoritmo *Rijndael*, eleito como *Advanced Encryption Standard* (AES), opera com blocos de 128 *bits* e chaves de 128, 192 ou 256 *bits*. Ao contrário do algoritmo DES e seus derivados, o AES não utiliza iterações em rede de funções Feistel; em vez disso, faz uso de uma rede de permutação-substituição. Ele foi projetado para ser mais eficiente em aplicações com baixo poder de processamento. Ele não possui uma quantidade de *rounds* fixa; para chaves de 128 *bits*, são realizados 10 *rounds*, para 192 *bits*, 12 *rounds* e para chaves de 256 *bits*, 14 iterações são realizadas [Daemen and Rijmen 2002].

No início do processo, o texto puro é disposto em matrizes de 4x4 *bytes* chamadas de estados. Em seguida, a chave é escalonada em sub-chaves de 128 *bits* cada, com a quantidade variando de acordo com o número de *rounds* e sendo uma chave por *round* mais uma adicional para a inicialização do processo. Em cada iteração, uma sub-chave é adicionada, depois é realizada uma etapa de substituição não linear dos *bytes*, de acordo com um estado tido como referência. Na sequência, acontece um deslocamento (*shift*) das linhas dos estados, e os *bytes* de cada coluna dos estados são transformados linearmente. O *round* final se consiste na realização da substituição não linear, transposição das linhas do estado e adição da última sub-chave [Daemen and Rijmen 2002]. A técnica é representada no pseudo-código 1.

Algoritmo 1 Advanced Encryptation Standard (AES)

```
1: função AES(State, CipherKey)
2:     KeyExpansion(CipherKey,ExpandedKey);
3:     AddRoundKey(State,ExpandedKey[0]);
4:     para keys K_i em CipherKey faça
5:         Round(State,ExpandedKey[K_i]);
6:     fim para
7:     FinalRound(State,ExpandedKey[K_f]);
8: fim função
9:
10: função ROUND(State, ExpandedKey[K_i])
11:     SubBytes (State);
12:     ShiftRows(State);
13:     MixColumns(State);
14:     AddRoundKey(State,ExpandedKey[K_i]);
15: fim função
16:
17: função FINALROUND(State, ExpandedKey[K_i])
18:     SubBytes (State);
19:     ShiftRows(State);
20:     AddRoundKey(State,ExpandedKey[K_i]);
21: fim função
```

2.2.6. Rivest-Shamir-Adleman (RSA)

Diferentemente dos algoritmos citados previamente, o RSA é dito de criptografia assimétrica, na qual as chaves privadas não são compartilhadas. Tal abordagem é bem difundida em aplicações que realizam transmissão de dados utilizando canais tidos como inseguros como, por exemplo, a Internet. O algoritmo é comumente utilizado para criptografar *e-mails*, autenticação de usuários, transações em sites de comércio eletrônico, aplicações de bate-papo, dentre outros serviços *online*.

A encriptação de mensagens c por meio do RSA é realizada calculando a exponenciação modular da mensagem m, que deve ser obrigatoriamente menor que $n - 1$, com o componente e da chave pública: $m^e \equiv c \times mod(n)$. Já a decriptação da mensagem criptografada é realizada com o mesmo processo, porém, utilizando o componente d da chave privada: $c^d \equiv m \times mod(n)$.

Para o processo de geração do par de chaves (pública e privada) são escolhidos pseudo-aleatoriamente dois números probabilisticamente primos (p e q) com quantidade de dígitos na ordem de 10^{100}. Estes números são multiplicados um pelo outro, produzindo n. Em seguida, é calculada a função Carmichael $\phi_{(n)}$ (cálculo do mínimo múltiplo comum entre os antecessores de p e q). Feito isso, um inteiro pseudo-aleatório e maior que 1 e menor que $\phi_{(n)}$ é escolhido de tal forma que sejam primos entre si. Este inteiro é usado no cálculo de d, de modo que d seja o inverso multiplicativo de e (ou seja, $d \times e \equiv 1$). Ao fim deste processo, dois conjuntos serão gerados: o que representa a chave privada (*PVK*) – par $[n, e]$ – e o que caracteriza a chave pública (*PBK*) – $[p, q, d]$. Este processo é ilustrado no diagrama da Figura 6.

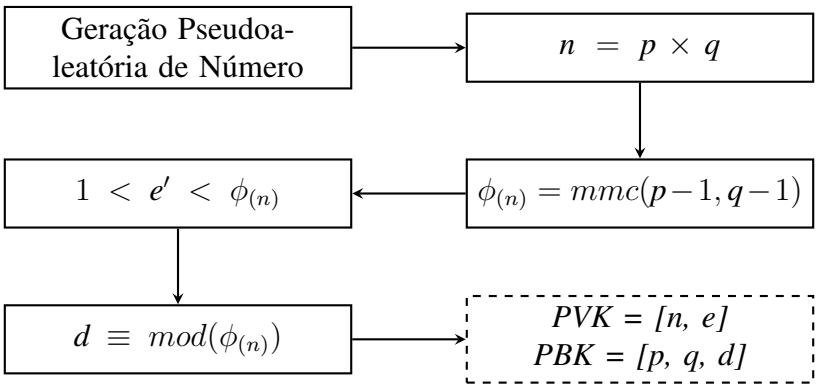

Figura 6. Diagrama de geração de chaves RSA.

2.3. Chaves Criptográficas

Chave criptográfica é uma informação que controla os algoritmos de criptografia. Por exemplo, para um mesmo algoritmo criptográfico e um mesmo texto puro, obtêm-se textos criptografados distintos ao utilizar chaves diferentes.

Existem basicamente dois tipos de chaves: as públicas e as privadas. As do primeiro tipo são comumente usadas em algoritmos de criptografia assimétricos, sendo responsáveis pela autenticação e encriptação da mensagem. Já as do segundo, além de serem usadas na criptografia assimétrica para decriptação da mensagem, também se fazem úteis

na criptografia simétrica, onde são compartilhadas pelo transmissor e receptor de tal forma que somente ela seja capaz de encriptar e decriptar uma mensagem.

Para que as chaves possam ser trocadas entre serviços, existem algoritmos que possibilitam estas transferências em meio a canais inseguros de comunicação. Um destes algoritmos de intercâmbio de chaves criptográficas é o Diffie-Hellman padronizado pela RFC2631 [Rescorla 1999].

O protocolo baseia-se em operações logarítmicas discretas para realizar manipulações locais com uma chave pública comum a ambas partes, de forma que uma nova informação comum secreta possa ser obtida ao final do processo. Entretanto, para realizar a troca pelo algoritmo Diffie-Hellman, é necessária a geração prévia de uma chave pública de uma chave privada. O protocolo está representado na Figura 7.

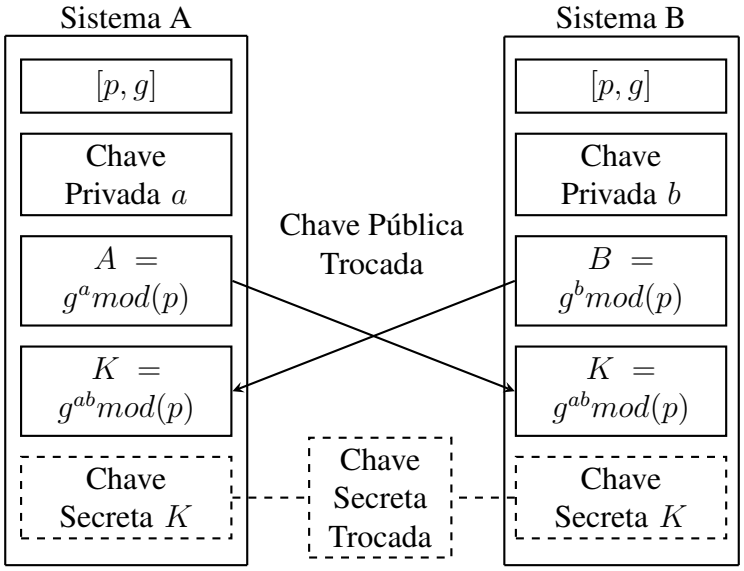

Figura 7. Diffie-Hellman para troca de chaves em meio a canais inseguros.

A segurança das chaves criptográficas são de grande importância para a segurança da informação, uma vez que se a chave for extraviada, os dados protegidos poderão ser descifrados por qualquer sistema que tenha o conhecimento das chaves envolvidas na criptografia. Como a grande maioria dos algoritmos criptográficos são de código aberto, a única forma de garantir a segurança das informações é mantendo o sigilo das chaves.

Um dos ataques mais simples para encontrar uma chave criptográfica é o chamado "força bruta" (*brute force*). Este ataque se consiste em realizar uma busca exaustiva em todas as possibilidades de combinações de caracteres até que a correta (chave) seja descoberta. O método possui a vantagem de ser 100% eficaz, ou seja, há garantia de que, cedo ou tarde, a chave secreta será descoberta, uma vez que ela pertence ao conjunto de todas as combinações possíveis. Porém, este tipo de ataque passa a ser computacionalmente inviável quando a chave em questão é longa, pois quanto maior a chave, maior o número de combinações possíveis [Stinson 2002]. Tal grandeza C cresce exponencialmente na razão $C = 2^n$ para chaves binárias, como é mostrado no gráfico da Figura 8.

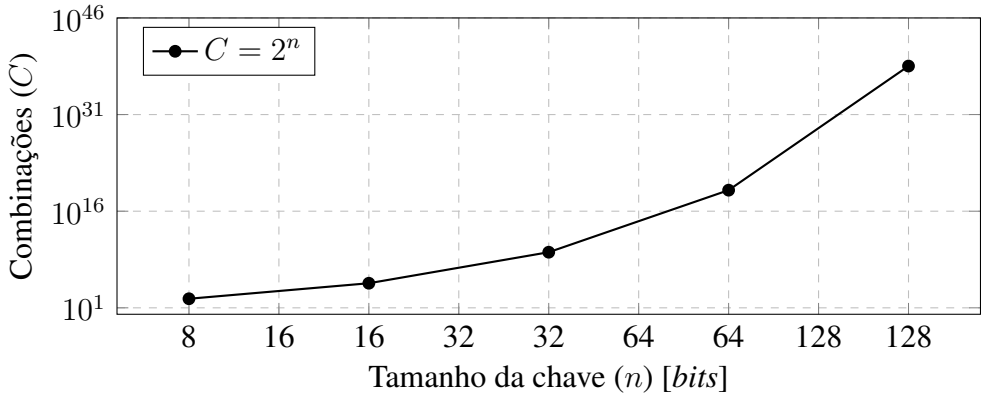

Figura 8. Quantidade de combinações possíveis para chaves binárias.

Um processo de geração de chaves criptográficas 100% aleatório garantiria que nenhum sistema consiga reproduzir esta chave por meio da repetição de instruções. Como a aleatoriedade perfeita é computacionalmente utópica, a pseudoaleatoriedade descrita pela RFC4086 é adotada neste tipo de sistema [Eastlake et al. 2005].

A pseudoaleatoriedade faz uso de parâmetros definidos externamente, que recebem o nome de entropia [Ristenpart and Yilek 2010]. Se consistem em informações que chegam a ser tão incertas que podem ser consideradas "aleatórias". Esta entropia é normalmente capturada em ações realizadas pela sistema como um todo como, por exemplo, movimentos do *driver* de disco, dentre outros eventos probabilísticos considerados imprevisíveis [Young and Yung 2004].

A entropia coletada do sistema é a base para a geração da maioria das chaves criptográficas. Tal fato implica em uma relação diretamente proporcional com o tamanho da chave e, consequentemente, com sua segurança. Tal necessidade pode ser um problema para sistemas reduzidos que não possuem variadas fontes de entropia como, por exemplo, sistemas embarcados, tornando assim a geração de números pseudoaleatórios ineficiente.

2.4. 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].

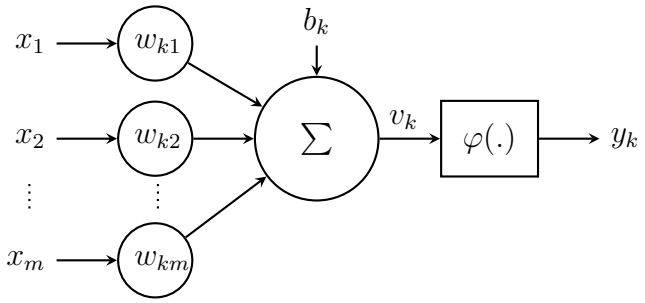

Figura 9. Representação de um neurônio artificial.

Os neurônios artificiais, representados pela Figura 9, 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 cujo as saídas podem ser conhecidas (treinamento supervisionado) ou não (treinamento não-supervisionado), onde cada iteração é uma época de treinamento. Uma regra de aprendizado é aplicada para atualizar os pesos sinápticos até que um critério de parada seja satisfeito. O conhecimento adquirido pela rede são os valores finais de seus pesos sinápticos.

Uma RNA pode possuir múltiplas camadas; neste caso, ela é denominadas *multilayer*. Quando a saída dos neurônios artificias presentes nas camadas intermediárias das redes não realimentam neurônios das camadas anteriores, as redes são classificadas como "de alimentação adiante" (*feedforward*).

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 feedforward* frequentemente utilizado em aplicações que envolvem criptografia neural (vide seção 2.5).

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 10 apresenta uma *Tree Parity Machine* parametrizada com $K = 3$ e $N = 4$.

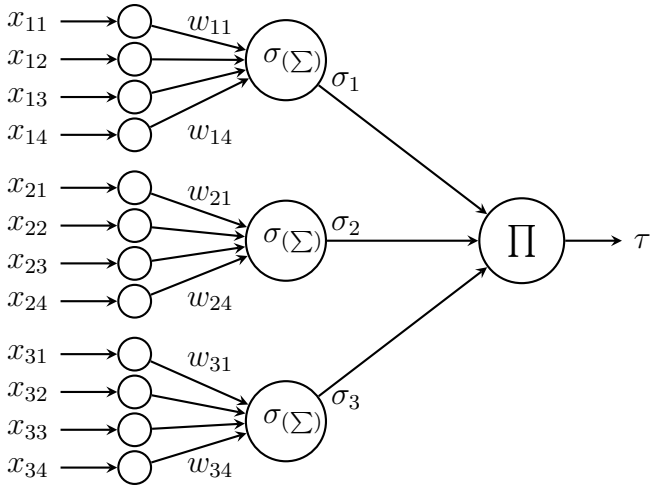

Figura 10. *Tree Parity Machine* (TPM) parametrizada com *K* = 3 e *N* = 4.

2.5. 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 2.4).

A técnica de geração de chaves utiliza da sincronização mutua entre duas *Tree Parity Machines* (representada pela Figura 11) 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.

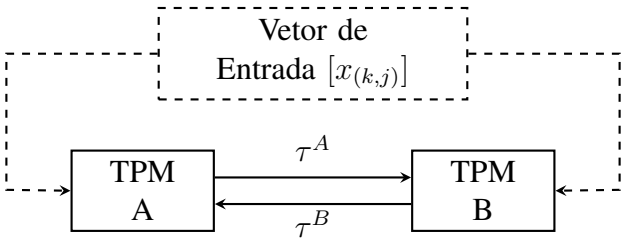

Figura 11. Sincronização entre *Tree Parity Machines*.

As redes somente atingem a sincronização quando seus vetores de pesos sinápticos (explicadas adiante na seção 2.4) 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.

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.

Um ponto crucial ao processo de geração de chaves criptográficas é a fonte de entropia. Monrose et al. [Monrose et al. 2001] e Chang et al. [Chang et al. 2004] propuseram o uso de características biométricas – captura de voz e captura facial – como fonte de entropia da pseudoaleatoriedade necessária, por serem consideradas únicas em seres vivos. Também foram consideradas o uso de impressões digitais, identificação de iris e geometria da mão. Tal abordagem pode tornar o procedimento de geração de chaves mais aleatório mas, ao mesmo tempo, estreitamente dependente de entradas externas.

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.

Piazentin [Piazentin 2011] realizou um estudo fortemente baseado na proposta de Ruttor [Ruttor 2006] sobre parâmetros de sincronização entre duas *Tree Parity Machines*. Em seu trabalho foram analisadas as influências dos parâmetros L, K, N e da regra de aprendizado no processo de sincronização com relação à segurança do proposta. Constatou-se que a regra de aprendizado que se mostrou mais eficiente em sua aplicação foi a *Hebbian* [Hebb 1949]. Como o próprio autor afirma, o desempenho não pôde ser aferido de fato, uma vez que o trabalho fora realizado em Python, linguagem de programação com alto nível de abstração. Também não foi possível definir uma configuração ideal dos parâmetros da rede.

No trabalho de Revankar et al. [Revankar et al. 2010] foi proposto um mecanismo de troca de chaves criptográficas utilizando criptografia neural, onde os vetores pseudoaleatórios de entrada são substituídos por consultas que dependem do estado atual da rede neural. Para isso, os autores utilizaram a regra descrita pela Equação (1), na qual $c_{k,\pm l}$ representa o número de produtos, sendo que o vetor de psos sinápticos ($w_{k,j}$) multiplicado pelo vetor de entrada ($x_{k,j}$) resulta em um parâmetro l ($w_{k,j} \times x_{k,j} = l$). Os autores concluíram que a sincronização realizada com as consultas para formação dos vetores de entrada foi mais rápida, porém, a quantidade de informação trocada entre as *Tree Parity Machines* foi maior. Portanto, embora o tempo de sincronização seja menor, o tempo total da geração de chave pode ser maior do que a geração por meio de sincronização com entradas pseudoaleatórias.

$$h_k = \frac{1}{\sqrt{N}} \times \sum_{l=1}^{L} (c_{k,+l} - c_{k,-l}) \tag{1}$$

Com o intuito de aumentar a segurança no procedimento de troca de chaves utilizando criptografia neural, Allam e Abbas [Allam and Abbas 2010] propuseram uma técnica na qual transmissões com conteúdo falso são realizadas durante o processo de sincronização entre as *Tree Parity Machines* (aqui chamadas de A e B). Durante a sincronização, são enviadas mensagens errôneas – baseadas nas distâncias estimadas entre os pesos sinápticos das redes neurais – descritas pela Equação (2). Isto faz com que uma terceira TPM (C), que possa estar interceptando pacotes com trocas de mensagens entre A e B (como em um ataque *Man-in-the-Middle*[10], por exemplo), não consiga se sincronizar e, consequentemente, obter a chave que será gerada ao final do processo.

$$\left| h_k^{A/B} \right| = \left| \sum_{n=1}^{N} (w_{k,n}^{A/B} \times x_{k,n}^{A/B}) \right| \tag{2}$$

Este trabalho objetivou a otimização dos parâmetros das TPMs utilizadas pela criptografia neural no procedimento de geração de chaves criptográficas. Também, propôs

[10] Ataque de interceptação de dados trocados entre duas partes.

uma implementação da técnica na linguagem de programação C[11] visando maior desempenho e tornando possível a comparação dos resultados obtidos pelas métricas do processo descrito na seção 4.

4. Proposta

A proposta deste trabalho, descrita pelo diagrama da Figura 12, 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 12 para modularizar a implementação estão descritos detalhadamente a seguir:

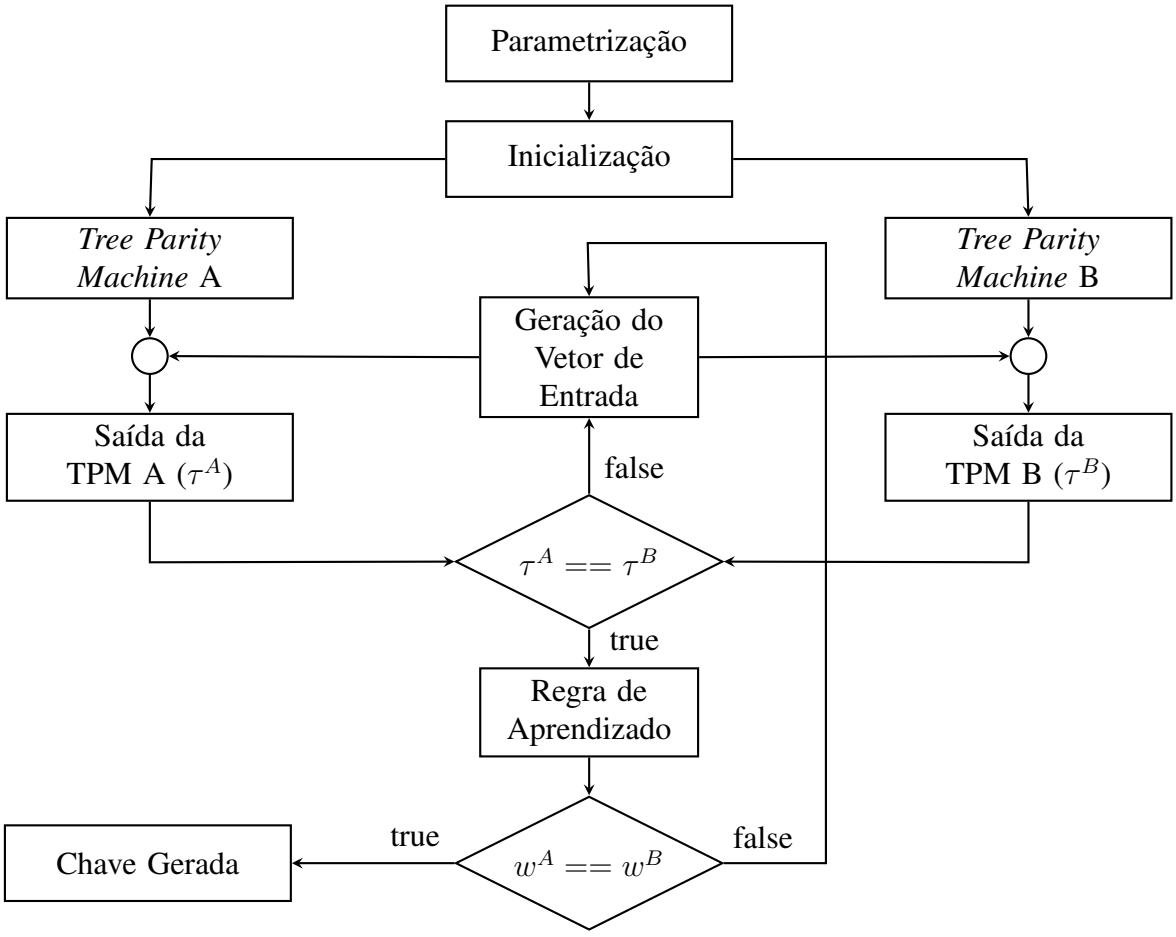

Figura 12. Diagrama em blocos da proposta.

- 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.

[11]Linguagem de programação compilada de uso geral criada em 1972 por Dennis Ritchie.

- *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 (3), 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.

$$\tau^{A/B} = \prod_{k=1}^{K} \sigma_k^{A/B} = \prod_{k=1}^{K} \sigma \left(\sum_{n=1}^{N} w_{k,n}^{A/B} \times x_{k,n}^{A/B} \right) \tag{3}$$

- 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 (4), *Anti-Hebbian* pela Equação (5) e *Random Walk* descrito pela Equação (6), 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.

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

$$W_{k,n}^{A/B} = w_{k,n}^{A/B} - x_{k,n}\sigma_k^{A/B}\Theta(\tau^{A/B}\sigma_k^{A/B})\Theta(\tau^A\tau^B) \tag{5}$$

$$W_{k,n}^{A/B} = w_{k,n}^{A/B} + x_{k,n}\Theta(\tau^{A/B}\sigma_k^{A/B})\Theta(\tau^A\tau^B) \tag{6}$$

$$\Theta(x) := \begin{cases} -1 & \text{if } x \leq 0 \\ +1 & \text{if } x > 0 \end{cases} \tag{7}$$

- 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 extrair o maior desempenho possível sobre 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, assim sendo capaz de obter os parâmetros ótimos das RNAs.

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:

$$\varphi(x) := \begin{cases} -1 & \text{if } x \leq 0 \\ +1 & \text{if } x > 0 \end{cases} \tag{8}$$

   (b) Sigmóide:

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

   (c) Linear:

$$\varphi(x) = x \tag{10}$$

   (d) Tangente Hiperbólica (tanh):

$$\varphi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{11}$$

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 (12).

$$P_k[\%] = \frac{1}{2^{(K \times N)}} \times 100 \tag{12}$$

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 (13). Bastante relevante para analisar situações onde as duas *Tree Parity Machines* não se encontram no mesmo sistema, ou até mesmo na mesma aplicação.

$$V[\textit{Bytes}] = \frac{\textit{epochs\_avg} \times [(K \times N) + K + 2]}{8} \tag{13}$$

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*).

$$I_{[K,N]} = (K \times N) + K \tag{14}$$

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 que haja coesão nas métricas temporais, todos os experimentos foram executados na mesma máquina cujo a configuração é:
   - Sistema Operacional: Xubuntu
   - Processador: Intel(R) Core(TM) i7-5500U
   - Clock Rate: 2.40 GHz
   - Arquitetura: x86_64
   - Núcleos: 4
   - Memória RAM: 16 GB

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 13.

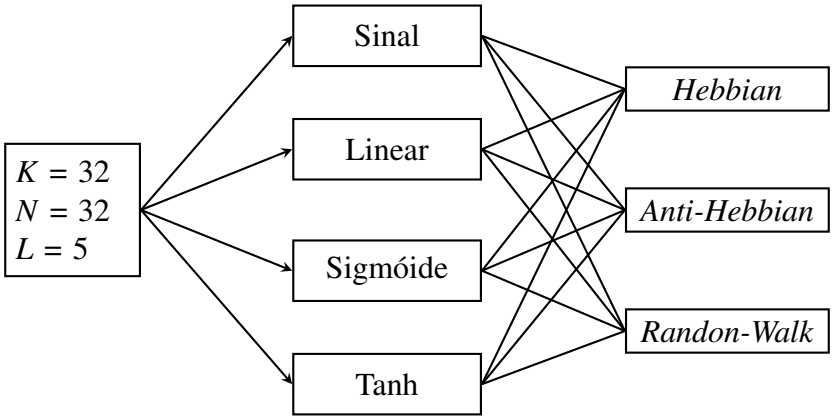

Figura 13. Experimento 1.

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 14. 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.

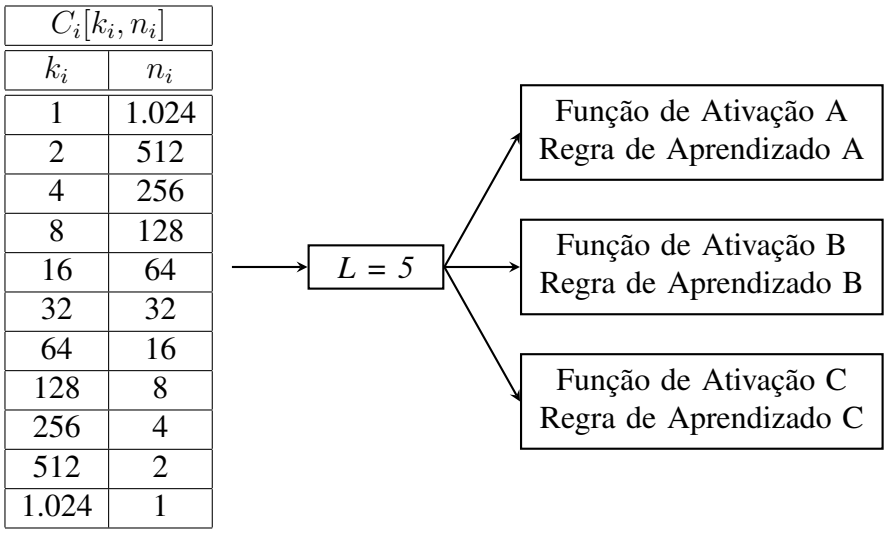

Figura 14. Experimento 2.

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 15, foi denominada "Experimento 3".

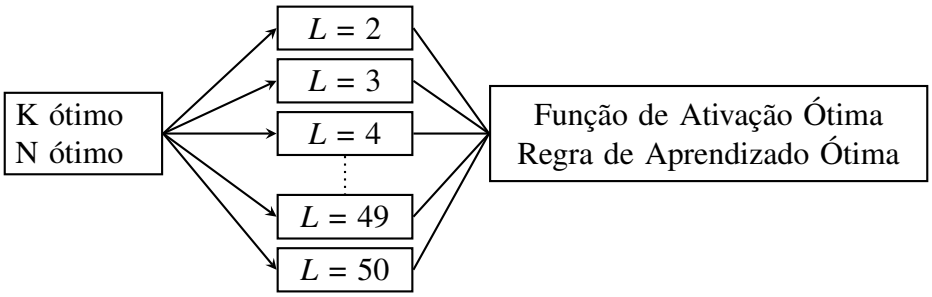

Figura 15. Experimento 3.

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

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 13), 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.

Para ilustrar os resultados, as métricas coletadas são dispostas na Tabela 2 e visualmente representadas nos gráficos das Figuras 16, 17 e 18. Logo em seguida os resultados são comentados.

Tabela 2. Tabela de resultados do Experimento 1.

Função de Ativação	Regra de Aprendizado	Número de Épocas		Tempo [microseg]		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

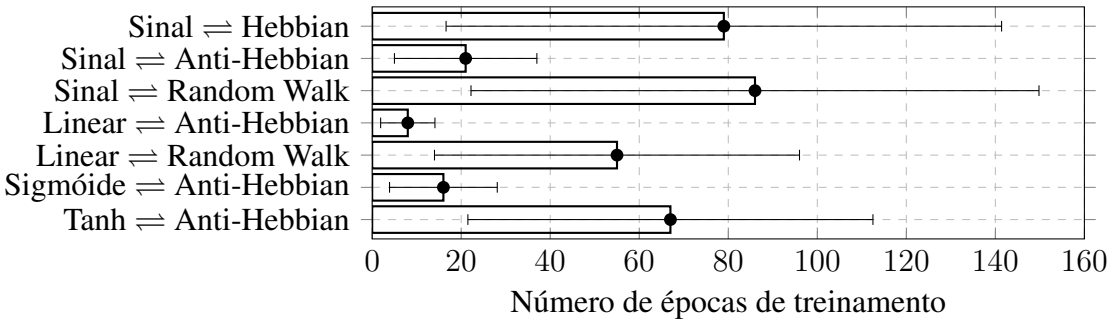

Figura 16. Número médio de épocas com desvio padrão.

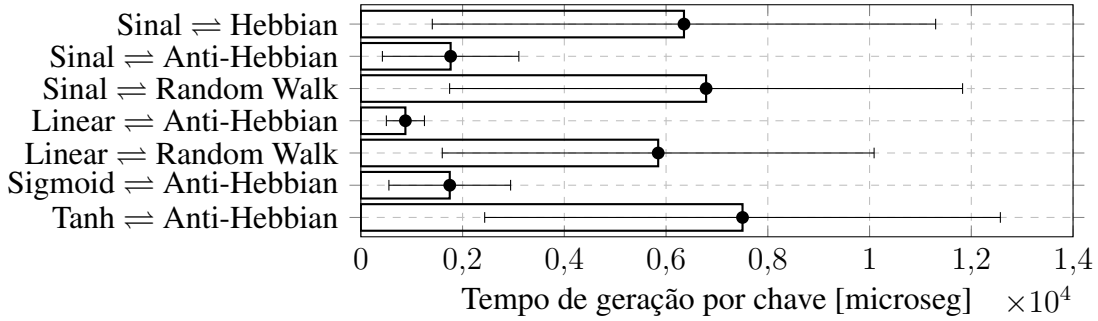

Figura 17. Tempo médio total de geração por chave com desvio padrão.

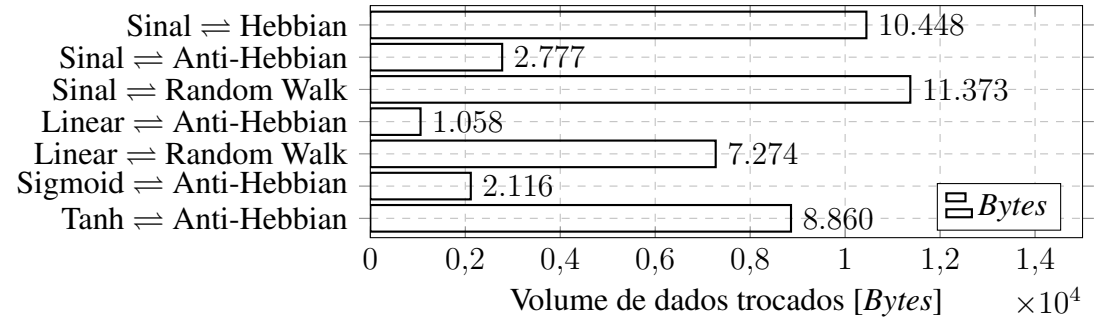

Figura 18. Volume médio de dados trocados por chave.

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

Foi constatado que o conjunto [sinal, *Anti-Hebbian*] se mostrou ótimo para esta implementação. Isto se deu ao fato do conjunto apresentar o menor tempo de geração por chave, 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 resultados estão dispostos na Tabela 4 e nos gráficos das Figuras 19, 20 e 21.

Tabela 4. Tabela de resultados do Experimento 2.

$C_i[k_i, n_i]$		Épocas		Tempo [microseg]		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

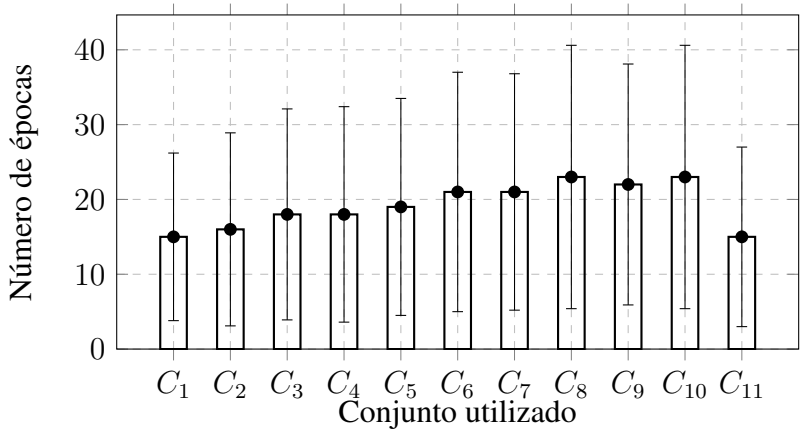

Figura 19. Análise do número de épocas com desvio padrão.

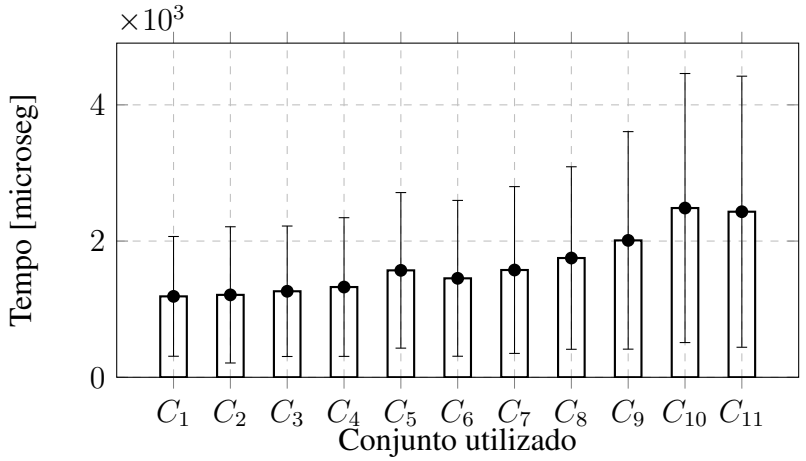

Figura 20. Análise do tempo de geração com desvio padrão.

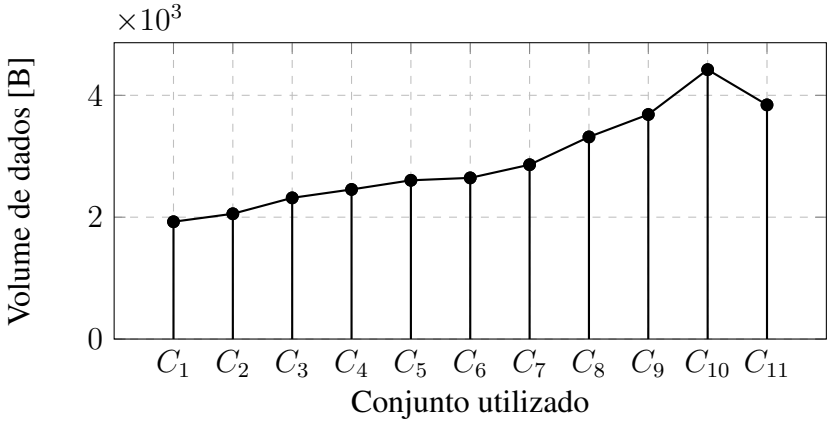

Figura 21. Análise de volume de dados.

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 (15) e (16).

$$I_{[1,1.024]} = (1.024 \times 1) + 1 = 1.025 \tag{15}$$

$$I_{[1.024,1]} = (1 \times 1.024) + 1.024 = 2.048 \tag{16}$$

A mesma conduta é identificada no cálculo do volume de dados. Esta está representada nas Equações (17) e (18).

$$V_{[1,1.024]} = \frac{15 \times [(1 \times 1.024) + 1 + 2]}{8} = 1.925[B] \tag{17}$$

$$V_{[1.024,1]} = \frac{15 \times [(1.024 \times 1) + 1.024 + 2]}{8} = 3.843[B] \tag{18}$$

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 22.

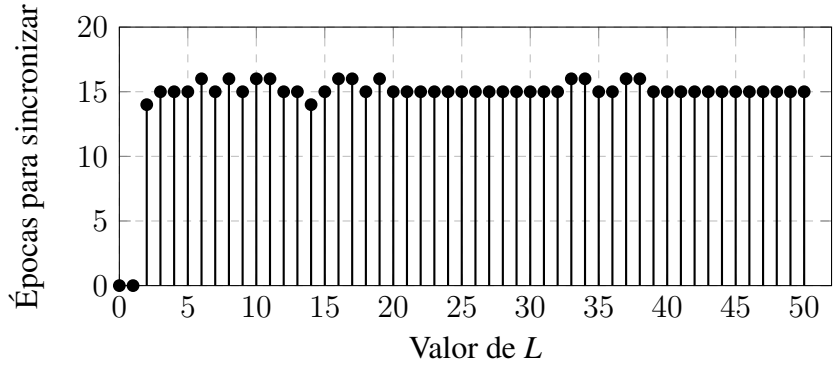

Figura 22. Análise da influência de *L* nas condições alcançadas.

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*

Com a configuração alcançada, os dois grupos de chaves propostos pelo Experimento 4 foram geradas, tal qual a coleta das métricas estudadas. Seus resultados foram gerados e estão representados pela Tabela 5 e pelos gráficos das Figuras 23 e 24.

Tabela 5. Tabela de resultados do Experimento 4.

Configuração Ótima	Número de Chaves	Número de Épocas		Tempo [microseg]		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

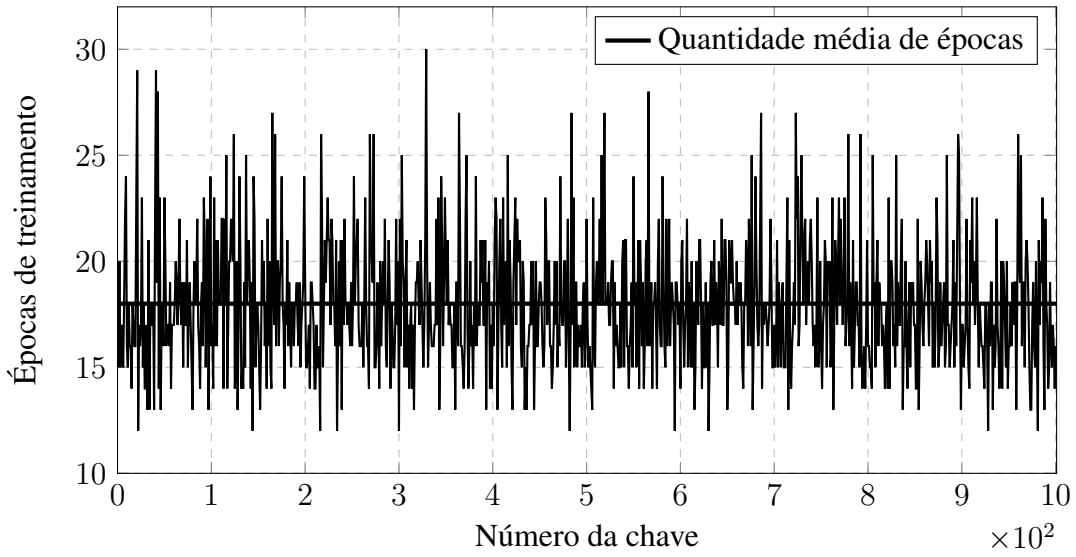

Figura 23. Épocas de treinamento na geração das 1.000 chaves binárias.

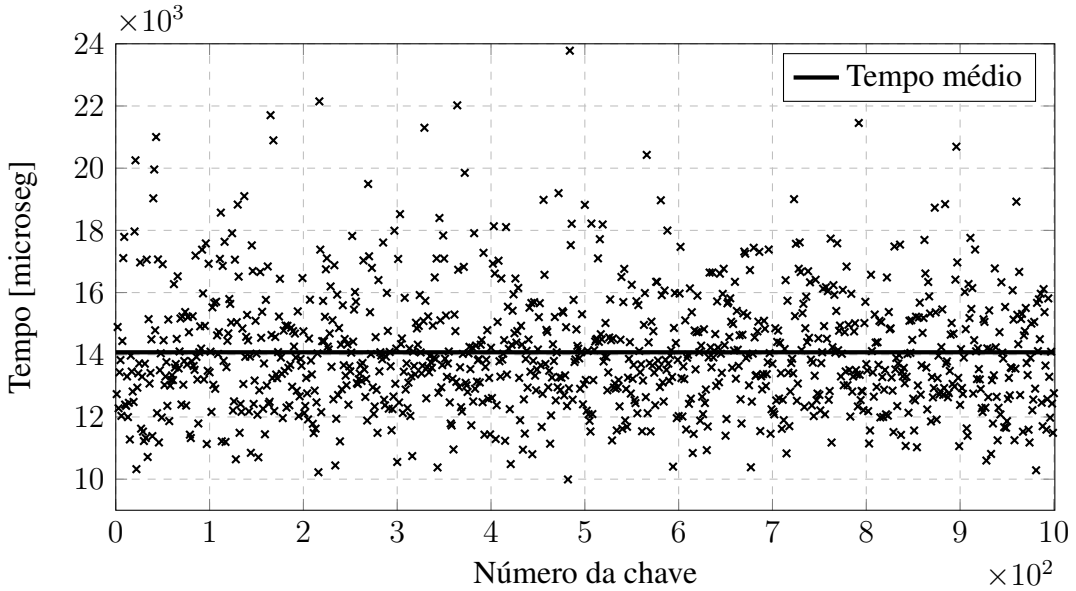

Figura 24. Tempo por chave de geração das 1.000 chaves binárias.

6. Conclusão

A partir da revolução industrial, o patrimônio intelectual vem sendo, cada vez mais, a propriedade mais valiosa no contexto empresarial. A ponto de que para a esmagadora maioria das instituições ativas, perder suas informações acarretaria em consequências calamitosas. Isto faz com que este bem tão precioso seja alvo dos afamados crimes cibernéticos, ataques que visam roubar ou inviabilizar estes dados.

Neste contexto surge a segurança da informação com a finalidade de proteger estes dados. A SI têm como uma das principais ferramentas, a criptografia, que possibilita a troca de informações entre partes por meio de canais inseguros de comunicação. Inúmeros algoritmos de criptografia foram e são propostos visando proteger as informações de formas cada vez mais eficientes. Como os algoritmos mais populares e utilizados são *open source*, é possível concluir que a segurança está na chave que é utilizada para encriptar os dados em suas utilizações.

Estas chaves são normalmente geradas com base na entropia (eventos probabilísticos considerados pseudo-aleatórios) coletada do sistema. A necessidade de uma fonte abundante de entropia pode ser um problema para sistemas reduzidos, como por exemplo em sistemas embarcados. Além da geração da chave, a troca da mesma entre serviços pode levar tempo e ser um ponto de ataque para que um terceiro obtenha uma cópia.

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. Os principais ataques são:

- Ataque simples [Ruttor et al. 2006]
- Ataque geométrico [Klimov et al. 2002]
- Ataque de maioria [Ruttor 2006]
- Ataque genético [Ruttor et al. 2006][Klimov et al. 2002]

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

Adams, C. (1997). The cast-128 encryption algorithm. Technical report, Network Working Group.

Adams, C. and Gilchrist, J. (1999). The cast-256 encryption algorithm. Technical report, Network Working Group.

Adshead, Antony (2017). Analytics, internet of things to drive data volumes to 163ZB by 2025. *Computer Weekly*. Disponível em <https://bit.ly/2qE9lS1>. Acessado em nov 2018.

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.

Allam, A. M. and Abbas, H. M. (2010). On the improvement of neural cryptography using erroneous transmitted information with error prediction. *IEEE Transactions on Neural Networks*, 21(12):1915–1924.

Antunes, M. T. P. and Martins, E. (2002). Capital intelectual: verdades e mitos. *Revista Contabilidade & Finanças*, 13(29):41–54.

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

Chang, Y.-j., Zhang, W., and Chen, T. (2004). Biometrics-Based Cryptographic Key Generation. In *International Conference on Multimedia and Expo (ICME)*, volume 3, pages 2203–2206. IEEE.

Cohen, F. (1995). A short history of cryptography, 1995. *URL: http://web. itu. edu. tr/˜ orssi/dersler/cryptography/Chap2-1. pdf (2017-09-17)*. Disponível em <http://all.net/edu/curr/ip/Chap2-1.html>. Acessado em nov 2018.

Daemen, J. and Rijmen, V. (2002). *The design of AES- the Advanced Encryption Standard*. Springer Science & Business Media.

Eastlake, D., Schiller, J., and Crocker, S. (2005). Randomness Requirements for Security. *NTWG - Network Working Group*. Disponível em <https://bit.ly/2DvvLNy>. Acessado em nov 2018.

Grabbe, J. (1992). The DES algorithm illustrated. *Laissez Faire City Times*, 2(28):1–15.

Hebb, D. O. (1949). *The Organization of Behavior; A Neuropsychological Theory*. Wiley & Sons, New York, New York, USA.

ISO/IEC (2013). Tecnologia da informação — Técnicas de segurança — Código de prática para controles de segurança da informação. *ABNT - Associação Brasileira de*

Normas Técnicas, page 99. Disponível em <https://bit.ly/2tXuIPd>. Acessado em nov 2018.

Klimov, A., Mityagin, A., and Shamir, A. (2002). Analysis of Neural Cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 288–298. Springer.

Lucks, S. (2001). The saturation attack—a bait for Twofish. In *International Workshop on Fast Software Encryption*, pages 1–15. Springer.

Masucci, B. (2003). Data encryption standard. *Federal Information Processing Standards Publication*.

Monrose, F., Reiter, M. K., Li, Q., and Wetzel, S. (2001). Cryptographic Key Generation from Voice. In *Proceedings of the {IEEE} Symposium on Research in Security and Privacy*, pages 202–212. IEEE.

Nakamura, E. T. and Geus, P. L. (2007). *Segurança de redes em ambientes cooperativos*. Novatec Editora.

Piazentin, D. R. d. M. (2011). Troca de chaves criptográficas utilizando criptografia neural. *UNIVEM*.

Redação (2014). 52% Das Empresas Brasileiras Tiveram Seus Dados Roubados Por Malware. *Canal Tech*. Disponível em <https://bit.ly/2Pl7v7o>. Acessado em nov 2018.

Redação (2018). Certificado digital do banco Inter é revogado após chave vazar na web. *G1 Globo*. Disponível em <https://glo.bo/2K0saI1>. Acessado em nov 2018.

Redação (2017a). Uber admite que omitiu ataque hacker que roubou dados de 57 milhões de usuários em 2016. *G1 Globo*. Disponível em <https://glo.bo/2zrOBna>. Acessado em nov 2018.

Redação (2017b). Volume de roubo de dados cresce 88% em 2017. *Computer World*. Disponível em <https://bit.ly/2Dvv8Uc>. Acessado em nov 2018.

Rescorla, E. (1999). Diffie-Hellman Key Agreement Method. Technical report, RFC2631.

Revankar, P., Gandhare, W., and Rathod, D. (2010). Private inputs to tree parity machine. *International Journal of Computer Theory and Engineering*, 2(4):665.

Riaz, M. and Heys, H. (1999). The FPGA implementation of the RC6 and CAST-256 encryption algorithms. In *Engineering Solutions for the Next Millennium. 1999 IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.99TH8411)*, volume 1, pages 367–372. IEEE.

Ristenpart, T. and Yilek, S. (2010). When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In *Ndss*. The Internet Society.

Rob, D. E. (1982). *Cryptography and Data Security*. Addison-Wesley Longman Publishing Co., Inc.

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

Ruttor, A., Kinzel, W., Naeh, R., and Kanter, I. (2006). Genetic attack on neural cryptography. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 73(3):36121.

Saint-Germain, R. (2005). Information Security Management Best Practice Based on ISO/IEC 17799. *Information Management*, 39(4):60–66.

Stallings, W. (2011). *Cryptography and Network Security*, volume 139. Pearson Education India.

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

Terada, R. (2008). *Segurança de dados criptografia em rede de computador*. Edgard Blucher.

Thakur, J. and Kumar, N. (2011). DES, AES and Blowfish: Symmetric key cryptography algorithms simulation based performance analysis. *International journal of emerging technology and advanced engineering*, 1(2):6–12.

Tolotti Umpieres, R. (2017). Corretora de bitcoin sul-coreana declara falência após ataque hacker. *Infomoney*. Disponível em <https://bit.ly/2PWLRG4>. Acessado em nov 2018.

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

Young, A. L. and Yung, M. (2004). *Cryptography: Malicious Cryptography – Exposing Cryptovirology*, volume 20. John Wiley & Sons.