UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Caio Bezerra Viana

Extração de Assinaturas de Identificação de Memória Flash Baseado em PUF com Arduino e Validação com Perceptron Multicamadas

Recife

2019

Caio Bezerra Viana

Extração de Assinaturas de Identificação de Memória Flash Baseado em PUF com Arduino e Validação com Perceptron Multicamadas

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação na Universidade Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Computação

Curso de Bacharelado em Ciência da Computação

Orientador: Carlos Julian Menezes Araújo

Recife

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

V614e Viana, Caio Bezerra
Extração de Assinaturas de Identificação de Memória Flash Baseado em PUF com Arduino e Validação com Perceptron Multicamadas / Caio Bezerra Viana. - 2019.
51 f.

Orientador: Carlos Julian Menezes Ara Araújo.
Inclui referências.

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Ciência da Computação, Recife, 2019.

1. Memória Flash. 2. Arduino. 3. Perceptron Multicamadas. 4. Weka. 5. PUF. I. Araújo, Carlos Julian Menezes Ara, orient. II. Título

CDD 004

MINISTÉRIO DA EDUCAÇÃO E DO DESPORTO
UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO (UFRPE)
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

http://www.bcc.ufrpe.br

FICHA DE APROVAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO

Trabalho defendido por Caio Bezerra Viana às 10:30 do dia 03 de dezembro de 2019, na Sala 42 do CEAGRI II, como requisito para conclusão do curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco, intitulado " **Utilização do Arduino para Identificação de Memória Flash Baseado em PUF com Perceptron Multicamadas** ", orientado por Carlos Julian Menezes Araújo e aprovado pela seguinte banca examinadora:

_____

Carlos Julian Menezes Araújo
DC/UFRPE

_____

George Gomes Cabral
DC/UFRPE

Este trabalho é inteiramente dedicado a minha mãe. A maior incentivadora das realizações dos meus sonhos. O meu maior exemplo. Muito obrigado.

Agradecimentos

A Engenheira Civil Rivete Bezerra Silva, minha mãe, por todos os conselhos que recebi durante a vida e seu apoio incondicional.

A Monja Coen Roshi, monja zen budista, por todos seus livros e palestras. Por auxiliar no meu crescimento pessoal, no equilíbrio físico e mental. E por me ensinar sobre as Quatro Nobres Verdades. Gasshô (mãos em prece).

Ao Professor Julian Menezes, meu orientador, por sua habilidade de compartilhar conhecimentos, por seu caráter, pelo suporte, pelas suas correções, pelos incentivos e por ser um ser humano exemplar.

A esta Universidade, seu corpo docente, direção, administração e pela ética aqui presente.

”A vida é um processo em si mesma. A morte é um processo em si mesma. Assim como a cinza não volta a ser lenha, a morte não volta a ser vida.”

(Mestre Zen Eihei Dogen Daiosho)

Resumo

A gestão de identidade de dispositivos é considerada um componente central para segurança na Internet das Coisas (*IoT*, do inglês, *Internet of Things*). Os principais métodos de autenticação usam o conceito de chave criptográficas, isso significa que a segurança fornecida pela criptografia está diretamente relacionada com a capacidade do sigilo dessa chave. Caso a chave seja conhecida por um atacante todo o processo de comunicação fica comprometido, visto que as mensagens podem ser decifradas. A fabricação de alguns dispositivos eletrônicos podem influenciar em seus comportamentos físicos, devido a existência de variáveis incontroláveis inerentes ao processo de fabricação. As técnicas PUF (do inglês, *Physical Unclonable Functions*) podem utilizar essas variáveis como fonte para geração de assinaturas de identificação de um *chip*. Este trabalho propõe uma abordagem de identificação de memória *flash*, motivado pela sua larga utilização nos dispositivos móveis atuais, que utiliza uma técnica de verificação de uma sequência de blocos da memória baseada na técnica PUF *Program Operation Latency*. Para isso, foi utilizada a plataforma Arduino como ferramenta para extração dessas assinaturas de identificação, em conjunto com a validação das assinaturas com o Perceptron Multicamadas (do inglês, *Multi Layer Perceptron* - MLP). O qual foi capaz de aprender o suficiente sobre essas assinaturas e generalizar no futuro, classificando corretamente as classes das assinaturas de identificação utilizadas nos testes, suportando dessa forma um mecanismo de diferenciação de memórias *flash*.

Palavras-chave: Memória *Flash*. Arduino. Perceptron Multicamadas. Weka. PUF.

Abstract

Device's identity management is considered a core component of IoT security. The main authentication methods use the concept of cryptographic key, this means that the security provided by encryption is directly related to the key's secrecy capacity. If this key is known by an intruder, the entire communication process is compromised since the messages' content can be decrypted. The manufacture of some electronic devices may influence their physical behavior due to the existence of uncontrollable variables inherent in the manufacturing process. Physical Unclonable Functions (PUF) techniques can use these variables as a source for generating chip's identification signatures. This work proposes a flash memory identification approach, due to the widespread use of this type of memory in current mobile devices, which use a memory block sequence verification technique based on the Program Operation Latency technique. With this aim, the Arduino platform was used as a tool for extracting these identification signatures, togheter with a validation of the signatures conducted by a MultiLayer Perceptron (MLP). Who was able to learn enough about these signatures and generalize in the future, and correctly classified the classes of identification signatures used in the tests, thus supporting a flash memory's differentiation mechanism.

Keywords: Arduino, Memory, Flash, Multilayer Perceptron, Weka.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

Arff	Attribute-Relation File Format
IC	Integrated Circuit
IoT	Internet of Things
MLP	Multi Layer Perceptron
PUF	Physical Unclonable Functions
SD	Secure Digital Card
USB	Universal Serial Bus
Weka	Waikato Environment for Knowledge Analysis

Sumário

1 Introdução

Este capítulo apresenta uma breve introdução sobre a importância da identificação dos dispositivos conectados em sistemas da Internet das Coisas, destacando limitações e decisões de projeto. Em seguida, são expostas a justificativa, o problema e os objetivos do trabalho proposto. Por fim, apresentamos a estutura do trabalho de conclusão do curso.

1.1 Contexto

Com o aumento da diversidade tecnológica, novas abordagens e protocolos têm sido criados com o objetivo de auxiliar a comunicação entre os novos equipamentos e os preexistentes. Houve também, um crescimento na quantidade de serviços que são utilizados nos sistemas da Internet das Coisas (do inglês, *Internet of Things* – *IoT*). Esses serviços podem usar uma enorme quantidade de dispositivos interconectados e necessitar da utilização de diversos protocolos para que ocorra um adequado funcionamento do sistema. Quando tratamos especificamente de dispositivos embarcados utilizados em ambientes *IoT*, vale ressaltar a presença de alguns desafios, por exemplo, a durabilidade das baterias utilizadas que podem limitar a capacidade do processamento do dispositivo (FERREIRA, 2014; HRIBERNIK et al., 2011).

Outro desafio que podemos considerar consiste na heterogeneidade dos dispositivos que são utilizados em um sistema *IoT*. Assim, o desenvolvimento de aplicações para esse tipo de sistema pode necessitar que a execução ocorra em diferentes plataformas, o que pode resultar em uma maior complexidade de integração. Visando minimizar esse tempo de desenvolvimento, alguns projetistas de sistemas *IoT*, consideram abstrair camadas inferiores, como ocorre com a utilização da plataforma *Arduino* (DOUKAS, 2012). Essa plataforma auxilia projetistas fornecendo a possibilidade de rápida prototipação, e possui a capacidade de receber novos sensores, atuadores e componentes que podem estar disponíveis como módulos padrões de *hardware* (ALMEIDA et al., 2015; FERREIRA, 2014; HRIBERNIK et al., 2011; SILVA; PRAZERES, 2013).

De uma maneira geral, podemos relacionar os requisitos de segurança para a *IoT* em alguns pontos: i) Gestão de identidade de usuários e dispositivos, que diz respeito ao processo de validação dos usuários para utilizar o sistema; ii) Confidencialidade dos dados trocados durante comunicação, que tem o objetivo de proteger os indivíduos envolvendo autenticação de comunicação em pares, para garantir confidencialidade e integridade dos dados comunicados; iii) Disponibilidade de recursos e sistemas, que diz respeito à limitação do poder computacional disponível nos dis-

positivos da *IoT*, que podem possuir uma capacidade de processamento insuficiente para execução de algoritmos de segurança; e iv) Controle de acesso à rede para garantir que a conexão à Internet seja disponível somente para dispositivos autorizados (BABAR et al., 2011; LIU; XIAO; CHEN, 2012).

Dentre esses pontos citados, a gestão de identidade de dispositivos é considerada um componente central para segurança na *IoT*, pois lida com a identificação e autenticação dos usuários e dispositivos em um sistema. E também é responsável por gerenciar os acessos aos recursos disponíveis através da vinculação de privilégios e restrições de acesso (BABAR et al., 2011; WANGHAM; DOMENECH; MELLO, 2013). Além disso, durante a autenticação dos dispositivos, a tecnologia e o conjunto de processos utilizados podem requerer diferentes tipos de métodos de autenticação. Contudo, diversas características podem influenciar na escolha do método utilizado, como por exemplo, os tipos de entidades envolvidas no sistema, e se os dispositivos podem pertencer a mais de uma rede ou domínio, etc. (WANGHAM; DOMENECH; MELLO, 2013).

Os principais métodos de autenticação usam o conceito de chave (GOODRICH; TAMASSIA, 2013). Este é um parâmetro necessário para execução de um algoritmo de criptografia, o qual utiliza a chave durante a comunicação para cifrar e decifrar mensagens impedindo assim que agentes externos, sem o conhecimento da chave, consigam decifrar essas mensagens (COLOURIS; DOLLIMORE; KINDBERG, 2007). Há duas grandes classes de algoritmos de criptografia em uso geral, a primeira remete a abordagem de utilizar chaves secretas, que são previamente conhecidas pelo remetente e destinatário, e não podem ser conhecidas por nenhum usuário. E a segunda abordagem lida com pares de chave pública/privada, na qual o remetente usa uma chave pública informada pelo destinatário para cifrar as mensagens, e o destinatário utiliza sua chave privada para decifrar essas mensagens (TANENBAUM, 2003). Independentemente da classe utilizada ocorre o armazenamento de uma chave, principalmente na perspectiva do dispositivo que visa utilizar a rede. Portanto, a segurança fornecida pela criptografia está diretamente relacionada com a capacidade de sigilo da chave utilizada no dispositivo, pois caso a chave seja conhecida por um dispositivo malicioso todo o processo de comunicação fica comprometido, visto que as mensagens podem ser decifradas (COLOURIS; DOLLIMORE; KINDBERG, 2007; WANGHAM; DOMENECH; MELLO, 2013).

Quando relacionamos a utilização de operações de autenticação em dispositivos eletrônicos, é importante salientar a presença significativa de recursos computacionais. Isso se deve, pois os circuitos integrados (do inglês, *Integrated Circuit - ICs*) necessitam de capacidade para executar tais operações. Caso contrário, as limitações dos ICs pode incapacitá-los de executar a operação ou de fazer com que até mesmo

a mais simples operação de criptografia seja demasiadamente custosa (SUH; DEVADAS, 2007). Quando se tratar de abordagens convencionais de autenticação, estas possuem algumas lacunas de segurança, como por exemplo, a necessidade de garantir o sigilo durante a manipulação de chaves criptográficas localizadas nas memórias desses ICs (SKOROBOGATOV, 2005). Particularmente, a memória *flash* é encontrada amplamente em diversos aparelhos eletrônicos como os *tablets*, *smartphones*, computadores, entre outros (SUH; DEVADAS, 2007).

Os *smartcards* são projetados para garantir confidencialidade e integridade de informações sensíveis, no entanto, enfrentam uma batalha contínua contra ataques. Ao mesmo tempo, os fabricantes dos *smartcards* tentam aumentar gradativamente a segurança em seus produtos, enquanto que atacantes tentam quebrar essas barreiras. Nesse sentido, os fabricantes dos *smartcards* não se responsabilizam caso seus produtos sejam invadidos (SKOROBOGATOV, 2005). Vale ressaltar que diversas formas de ataques as memórias têm sido desenvolvidas pela comunidade *hacker* (JING et al., 2014). Podemos citar como exemplo, os ataques físicos com a inserção de um dispositivo malicioso que é utilizado como uma ferramenta de espionagem, e que pode obter chaves criptográficas que estejam salvas nas memórias dos dispositivos (RESPONSE, 2014).

1.2 Justificativa

A área de autenticação e autorização de dispositivos em *Iot*, apesar de largamente abordada na literatura, ainda não apresenta acordos ou padrões comuns. (ZHANG et al., 2014). Da mesma forma, torna-se complexo padronizar protocolos, ou plataformas em sistemas *IoT*, devido a grande heterogeneidade de dispositivos que são utilizados. Esse aspecto tem influenciado o surgimento de interfaces específicas, ou até casos de protocolos reduzidos, para que seja possível a utilização de dispositivos de baixa capacidade de processamento (SKOROBOGATOV, 2005). Consequentemente, quanto maior o número de interfaces, protocolos ou plataformas maior será o custo para desenvolver o sistema em *IoT*.

Um sistema *IoT* só é considerado seguro se atender os objetivos de segurança para o qual foi desenvolvido (SANTOS et al., 2016). Vale ressaltar que, a complexidade e o tamanho de alguns protocolos criptográficos encarecem os projetos que utilizam segurança nesses sistemas. Caso os dispositivos embarcados sejam utilizados nesses sistemas, estes trazem consigo limites computacionais, por exemplo, a capacidade da bateria, o limite de processamento, a quantidade de memória disponível, entre outros (BABAR et al., 2011). Essas limitações motivam a criação de protocolos específicos que sejam capazes de executar nesse tipo de dispositivo. Outra motivação para a

alteração nos protocolos consiste na mitigação do excessivo aumento dos custos nos projetos de sistemas IoT. Porém, essas alterações nos protocolos podem influenciar no surgimento de falhas de segurança, que não são facilmente encontradas e corrigidas (SKOROBOGATOV, 2005).

Muitas aplicações computacionais requerem a capacidade de identificação única de um dispositivo. Por exemplo, os serviços relacionados a segurança ou que visam preservar direitos de fabricação (SUH; DEVADAS, 2007). Para isso, alguns pesquisadores sugerem a utilização técnicas baseadas em PUF (do inglês, *Physical Unclonable Functions*). A técnica PUF é uma função que, quando estimulada por um conjunto de desafios [1], gera um conjunto de respostas. Esse conjunto de respostas gerado é definido por propriedades complexas advindas do material físico onde se é utilizada, por isso a PUF é considerada uma função física (PRABHU et al., 2011). E pode ser utilizada como uma forma de identificação e autenticação de baixo custo (SUTAR; RAHA; RAGHUNATHAN, 2018).

1.2.1 Por que utilizar técnicas PUF ?

A identificação de dispositivos através do uso das técnicas PUF se mostraram uma alternativa plausível, pois são consideradas técnicas de segurança de baixo custo, que utilizam características físicas que são difíceis de serem replicadas pelos atuais processos de fabricação (SUH; DEVADAS, 2007). Além disso, essas técnicas podem ser adaptadas e utilizadas em memórias de dispositivos, componente este fundamental em qualquer computador. Além disso, podendo ser adotado para gerar assinaturas de identificação desses dispositivos (PRABHU et al., 2011).

Este trabalho tem como foco a adoção de uma técnica de identificação, baseada em técnicas PUF que utilizam a manipulação de características físicas da memória *flash* para dificultar o acesso as chaves criptográficas por atacantes. Os quais ocasionalmente podem tentar acessá-las através de ataques invasivos nas memórias *flash*. Vale ressaltar que, esse tipo de memória possui vulnerabilidades de segurança, pois garantir manipulação de chaves criptográficas nesse tipo memória é complexo e custoso (SUH; DEVADAS, 2007).

Quando se utiliza uma assinatura de identificação obtida através de uma técnica PUF, a manipulação e o armazenamento dessa assinatura só ocorreria na memória volátil do dispositivo. Dessa maneira, caso a alimentação elétrica do dispositivo fosse interrompida, como se é necessário para utilização de alguns tipos de ataques, a assinatura seria perdida devido a volatilidade da memória. Essa característica é utilizada como uma barreira a mais de segurança de *hardware* (MOROZOV; MAITI; SCHAUMONT, 2010).

[1] Challenge–response authentication

A técnica de identificação proposta nesse trabalho fundamenta-se em técnicas PUF, visto que, nos últimos anos pesquisas têm sido propostas na identificação de ICs considerando variações aleatórias do processo de fabricação de um dispositivo (TEHRANIPOOR et al., 2015). Porém, tais variações só podem ser utilizadas se forem capazes de gerar uma assinatura de identificação única. Essas assinaturas podem ser utilizadas como chaves de identificação, e não podem ser clonadas pelos métodos de fabricação atuais, sendo assim, extremamente difíceis de serem extraídas por terceiros (PRABHU et al., 2011).

Existem diversas técnicas baseadas em PUF, as quais em geral são específicas para cada tipo de dispositivo, ou ICs, que se pretende identificar. E como essas técnicas estão diretamente relacionadas às características físicas dos dispositivos os algoritmos utilizados são usualmente implementados em baixo nível para que seja possível analisar bit a bit. Quando se trata da utilização de técnicas PUF em memórias *flash*, podemos mencionar sete técnicas. Dentre essas técnicas, a técnica *Program Disturb* e a *Program Operation Latency* obteveram resultados satisfatórios quando implementas em FPGA para identificação da memória *flash* (SUH; DEVADAS, 2007).

1.2.2 Ataques Invasivos em Memória

Os Ataques podem ser usados de diferentes formas dependendo do objetivo, visando por exemplo o roubo de propriedade intelectual para clonar um dispositivo que está em alta no mercado com o intuito de se conseguir dinheiro fácil. Devido ao interesse de alguns competidores desonestos que visam reduzir custos em cima da cópia de produtos, a clonagem se tornou um ataque amplamente difundido (SKOROBOGATOV, 2005).

De forma geral os ataques invasivos em memória necessitam que o atacante tenha algum conhecimento prévio sobre o circuito do dispositivo que tentará invadir. Esse conhecimento é importante para que se possa conseguir realizar testes que são necessários para realizar o ataque, e assim conseguir os dados da memória. Entretanto, os testes necessitam que sejam executados em um laboratório altamente especializado para que os dados contidos na memória não sejam corrompidos durante esse tipo ataque(SKOROBOGATOV, 2005).

As técnicas PUF, como falado anteriormente, são utilizadas por sua capacidade de extrair uma assinatura de identificação de uma característica física do dispositivo (KELLER et al., 2014). Por exemplo, caso um algoritmo extraísse uma assinatura de identificação de um dispositivo através da memória *flash*, manteria essa assinatura na memória volátil do dispositivo para utilizá-la durante o fluxo de execução do algoritmo. O que obrigaria um atacante, que desejasse clonar essa assinatura, a manter a alimentação elétrica do dispositivo para que fosse possível efetuar um ataque invasivo,

pois caso contrário a cópia da assinatura de identificação PUF presente na memória volátil se perderia, junto com todos os dados dessa memória volátil depois da perda de alimentação elétrica. Contudo, um circuito de verificação de tensão poderia impedir esse tipo de tentativa, devido ao ruído causado na tensão durante o ataque (SKOROBOGATOV, 2005).

Existem diversos exemplos de ataques invasivos para técnicas PUF (SUTAR; RAHA; RAGHUNATHAN, 2018). Porém, como essas técnicas dependem diretamente do tipo do dispositivo, muitas vezes não é possível compará-las com outras técnicas por serem dispositivos diferentes. Entre os ataques relacionados a memória volátil vale ressaltar um exemplo encontrado na literatura, que demonstra um caso de sucesso utilizando uma implementação de uma técnica SRAM PUF (HELFMEIER et al., 2013).

Contudo, caso um atacante tente extrair a assinatura de identificação da memória volátil, durante o funcionamento de um dispositivo, o qual, possuiria a memória integrada a ele, o atacante assim necessitaria de uma tecnologia de alta precisão durante o processo. Pois, o atacante deveria possuir um equipamento de alta precisão, devido à necessidade de garantir que a execução do ataque não altere os *delays* da própria memória durante o processo (SKOROBOGATOV, 2005). Entretanto, se os *delays* forem alterados, a chave obtida não seria igual à chave criptográfica original (SUH; DEVADAS, 2007).

1.3 Problema de Pesquisa

Durante decisões de projetos de sistemas *IoT*, alguns procolos podem sofrer mudanças por causa dos limites computacionais. Tais modificações são ocasionadas pelos atuais dispositivos embarcados nos sistemas IoT (ver subseção 1.2.2). Essas mudanças podem gerar falhas de segurança que são dificilmente encontradas (SKOROBOGATOV, 2005). Adicionalmente, estas falhas podem ser utilizadas em ataques invasivos com intuito de tentar clonar qualquer chave criptográfica armazenada em memória *flash*. Essa chave pode ser usada para quebrar a segurança da comunicação, e caso clonada permite que, por exemplo, o atacante possa acessar informações privadas, ter acesso ou manipular recursos importantes do sistema ao simular ser um dispositivo legítimo (LIU; XIAO; CHEN, 2012).

Alguns pesquisadores utilizam as técnicas PUF como mecanismo para mitigar a vulnerabilidade da chave criptográfica armazenada em memória. Isso porque o processo para obtenção da assinatura de identificação das memórias dos dispositivos pode ser requisitado quando for necessário, por exemplo, com a utilização de um conjunto de desafios como foi sugerido por (TEHRANIPOOR et al., 2015). Desta forma, não obriga que exista uma chave criptográfica armazenada fisicamente na memória

flash de um dispositivo embarcado que deseje se comunicar. As técnicas PUF são baseadas na idiossincrasia de cada memória, ou seja, as características físicas oriundas do processo de fabricação são utilizadas para gerar as assinaturas de identificação. Contudo, vale ressaltar que essas idiossincrasias não podem ser clonadas pelo processo de fabricação de memórias atual, e que seria muito difícil de replicar (PRABHU et al., 2011). As assinaturas de identificação providas pela execução de técnicas PUF identificam unicamente uma memória, e por conseguinte, pode identificar um dispositivo que a utilize (TEHRANIPOOR et al., 2015).

Quando são considerados os tipos de ataques às memórias, os tipos de técnicas PUF e as citadas lacunas da segurança, relacionadas com a autenticação de dispositivos em IoT, reforça-se que a utilização de técnicas PUF para identificação de memória *flash* que são altamente utilizadas pelos dispositivos atuais é plausível. Nesse sentido, a proposta deste trabalho busca adotar uma técnica de identificação de blocos de uma memória *flash* como mecanismo para autenticação de dispositivos IoT.

1.4 Objetivo

Este trabalho tem como objetivo demonstrar uma abordagem de identificação de memória *flash*, mais especificamente, uma técnica de identificação de uma sequência de blocos baseada na técnica PUF, conhecida como *Program Operation Latency*. Através da utilização da plataforma Arduino, ocorrerá a extração de assinaturas de identificação, e após isso, a validação das assinaturas por meio do Perceptron Multicamadas (do inglês, *Multi Layer Perceptron* - MLP).

Dentre os objetivos específicos deste trabalho, apontam-se os seguintes:

- Definir um procedimento para analisar as assinaturas;
- Propor um algoritmo para extração e leituras de assinaturas de identificação;
- Implementar um circuito eletrônico com Arduino para extração de assinaturas de identificação de um cartão microSD.
- Adotar o uso de MLPs para validar as assinaturas de identificação.

1.5 Organização do trabalho

O presente trabalho encontra-se devido em cinco capítulos. Inicialmente, o primeiro capítulo apresenta a introdução, que compreende comentários iniciais, tema, problema, objetivo, assim como a justificativa. O capítulo 2, por sua vez, apresenta uma revisão teórica sobre os tipos de ataques às memórias, as técnicas de *physical unclonable functions* e o Perceptron Multicamadas. O capítulo 3 descreve a visão geral da metodologia, a configuração de ambiente, a extração de assinaturas de identificação, a formatação e o armazenamento de arquivos, a utilização da plataforma Weka, a avaliação dos blocos representativos da memória flash, e por fim, as métricas definidas por padrão no Weka. O capítulo 4 fornece o plano de experimento, e descreve a memória *flash*, a avaliação dos blocos representativos e o resultados encontrados. Por fim, o capítulo 5 descreve as conclusões, as limitações, as contribuições e indica os trabalhos futuros.

2 Fundamentação Teórica

Este capítulo está dedicado a fornecer uma revisão dos principais conceitos para auxiliar a compreensão e o desenvolvimento deste trabalho.

2.1 Tipos de Ataques às Memórias

Os detalhes sobre as informações relacionadas a proteção de segurança em microcontroladores e *smartcards* são escassos nos *datasheets*. Estes documentos geralmente fornecem apenas uma lista resumida de ataques, que foram testados contra a proteção de segurança utilizada pelo dispositivo, e quaisquer outros detalhes de implementação são usualmente suprimidos (SKOROBOGATOV, 2005). Além disso, o processo de armazenamento de chaves criptográficas possui alguns gargalos para IoT, tanto pela necessidade de proteção e sigilo das próprias chaves, quanto pela dificuldade de implementação de segurança devido aos já citados limites computacionais de processamentos de alguns circuitos integrados.

Os ataques às memórias de dispositivos podem ser divididos em duas categorias gerais (SKOROBOGATOV, 2005):

a) Ataques invasivos, que requerem horas ou semanas em laboratório especializado e durante o processo destroem o invólucro do IC, como por exemplo a Engenharia Reversa e o *Microprobing*.

b) Ataques Não Invasivos, os quais não manipulam o *hardware* fisicamente durante os ataques, por exemplo os Ataques de *Software* e *Eavesdropping*.

Vale ressaltar alguns desses tipos de ataques como, por exemplo, o *Microprobing* que são utilizados através do acesso direto aos ICs, onde se é possível observar, manipular e interferir nos circuitos integrados. Outro tipo de ataque invasivo e largamente conhecido é a Engenharia Reversa, a qual é utilizada para entender a estrutura interna do ICs e aprender ou emular suas funcionalidades, e para que isso ocorra pode até destruir o componente analisado. Um exemplo de ataque não invasivo são os Ataques de *Software* que utilizam a *interface* de comunicação normal com o processador e exploram as vulnerabilidades de segurança dos protocolos, dos algoritmos de criptografia e de suas implementações. Outro tipo de ataque é *Eavesdropping*, que são técnicas que permitem que os *hackers* monitorem, com alta taxa de resolução no tempo, as características analógicas de alimentação e das conexões de interface, ou qualquer radiação eletromagnética produzida pelo processador durante uma operação normal (SKOROBOGATOV, 2005).

Baseado nos ataques citados a necessidade de obtenção de uma impressão digital para identificação única de cada IC, que nenhum *hacker* possa obter ou duplicar, se tornou um dos grandes desafios da IoT (TEHRANIPOOR et al., 2015). Um alternativa encontrada por alguns pesquisadores são técnicas para a identificação dos ICs, baseadas em Physical Unclonable Functions (PUFs) que serão detalhadas no subtópico a seguir.

2.2 Técnicas de Physical Unclonable Functions

As primitivas criptográficas que derivam segredos proveniente de um conjunto complexo de características físicas de ICs são conhecidas como funções físicas não clonáveis (*Physical Unclonable Functions*, conhecidas como PUFs) (SUH; DEVADAS, 2007). Durante o processo de fabricação de um IC ocorrem variações aleatórias (como variações de *delay* nos fios ou transistores) que são extremamente difíceis de prever ou extrair, que podem ser utilizadas por técnicas PUFs para obtenção de uma chave. Algumas técnicas baseadas em PUFs foram propostas por pesquisadores com intuito de identificar unicamente memórias, o que foi chamado de impressão digital da memória (CAULFIELD et al., 2009; PRABHU et al., 2011; RAHMATI et al., 2016; WANG et al., 2012).

As idiossincrasias da memória *flash* causa mudanças de comportamento que ocorrem entre chips, suas origens estão relacionadas com as variações físicas existentes que são oriundas ao processo de fabricação. Essas idiossincrasias foram utilizadas como base para identificação única desse tipo de memória, através da obtenção de assinaturas, ou impressões digitais por técnicas PUFs. Essas assinaturas dependem diretamente das mudanças aleatórias no comportamento do dispositivo *flash* que foram detectadas através da interface elétrica. A memória *flash* é organizada de tal forma que um bloco (unidade básica do *array* da memória *flash*) pode ser apagado em uma única ação (STALLINGS, 2010). Cada bloco é compreendido como um conjunto de 16 ou 32 mil cadeias de bits em paralelo, onde cada célula de uma cadeia pertence a uma página diferente.

Vale salientar que durante o processo de fabricação as células da memória são colocadas tão próximas quanto for possível, com o intuito de maximizar a capacidade da memória. Por isso, pequenas variações geométricas, ou mesmo o processo de oxidação pode resultar em grande impacto no comportamento individual de cada célula, ou efeitos entre células adjacentes. Entre as diversas técnicas PUFs que foram definidas para extração de assinaturas únicas de memórias *flash*, destaca-se a *Program Disturb, Read Disturb e a Program Operation Latency* (PRABHU et al., 2011; CAULFIELD et al., 2009).

A técnica *Program Disturb* é caracterizada como uma técnica que extrai uma assinatura, ou impressão digital de uma memória, a qual é gerada a partir de iterações de um programa em uma página de um bloco da memória previamente apagado. Esta página é alterada repetidas vezes para que, entre cada iteração durante a execução do programa seja analisado se essa alteração induziu erros nas páginas fisicamente adjacentes. Se um erro no bit *n* localizado na página adjacente é encontrado em determinado número de iterações se é registrado o número da iteração que produziu o erro. Com base nisso, a assinatura é criada através da junção de todos os valores de erro no bit versus número de iterações necessárias para produzir esse erro, por exemplo a tupla (*e*, *n*) que representa um erro no bit *e* na enésima iteração do programa.

Outra técnica conhecida é a *Read Disturb* que gera a assinatura da memória *flash* baseada no erros induzidos pela operação de leitura. Essa técnica consiste em apagar um bloco inteiro da memória e depois o preencher com dados aleatórios, em seguida esse bloco é então submetido iterações de leitura de páginas de um bloco milhões de vezes para induzir um erro. Um exemplo de execução da técnica é ler o bloco após cada intervalo de 1000 iterações de leitura consecutivas, e checar a presença de erros nos bits de cada página no bloco. A junção da contagem de ciclos de iteração, com o bit e a página que ocorreu o erro é utilizado para gerar a assinatura, sendo o exemplo a tupla (*i*, *b*, *p*) que representa um erro no bit *i*, que está presente na página *p*, durante a iteração *b* do programa.

A técnica *Program Operation Latency* utiliza as varições entre as latências entre as operações da memória *flash*. Cada memória possui valores de latência para suas operações, devido a isso os fabricantes utilizam um range de números de ciclos de clock para a execução de algumas operações (TOCCI; WIDMER; MOSS, 2007), como por exemplo para que a memória execute a operação N é necessário entre 10000 a 50000 ciclos de clock. A assinatura é gerada ao se utilizar as variações das latências que ocorrem nas operações de gravação, como por exemplo, cada uma dessas operações gravando um bit por página durante a execução do programa.

Uma análise dessas técnicas citadas foi desenvolvida por (PRABHU et al., 2011), dentre as características relacionadas às técnicas PUFs, o tempo de retirada de assinatura e a correlação existente sobre assinaturas retiradas consecutivamente de um mesmo bloco são mostrados na tabela 1. Sendo necessário destacar que a técnica *Program Disturb* se demonstrou rápida quando comparada a *Read Disturb*, e possui uma correlação alta e com pouca variação quando comparada com a *Program Operation Latency*.

O conceito de PUF também pode ser aplicado em outros tipos de memórias, como por exemplo o caso da *dynamic random access memories* (Memória de Acesso

Tabela 1 – Comparação entre técnicas de extração de assinaturas únicas em memórias *flash*, com correlação de assinaturas de um dispositivo *multi-level cell*(MLC).

Técnica	Tempo (s)	Correlação
Program Disturb	100	0.85 - 0.90
Read Disturb	86400	0.98 - 1.00
Program Operation Latency	15-20	0.40 - 0.90

Fonte: (PRABHU et al., 2011)

Randômico Dinâmica, DRAM). As técnicas que utilizam esse tipo de memórias são conhecidas com DRAM PUFs, e podem ser usadas como uma aplicação de identificação de baixo custo, e também como gerador de chaves para criptografia. Contudo, a variação de temperatura ou da voltagem pode influenciar na estabilidade dos bits na memória DRAM, o que gera perda de informações (TEHRANIPOOR et al., 2015). Métricas para DRAM PUF foram desenvolvidas com a utilização de computação aproximada, e houve demonstrações que assinaturas extraídas durante diferentes iterações convergem para uma única, e que isso pode ser suficiente para identificar o dispositivo como uma impressão digital (RAHMATI et al., 2016).

2.3 Plataforma Arduino

O Arduino é uma plataforma de código aberto formada por uma junção de um ambiente de programação e placas eletrônicas com entradas e saídas, onde é possível que o desenvolvedor gerencie, compile códigos, realize *uploads*, e simule programas (SOLIMAN et al., 2013). A plataforma Arduino possui diversas placas com diferentes configurações. Isso possibilita que o desenvolvedor adquira a configuração de placa mais adequada ao seu projeto, podendo escolher, por exemplo, o tamanho de memória RAM, a taxa de *clock*, entre outros. O Arduino pode ser utilizado em conjunto com softwares de seu computador como, por exemplo, o *Processing* (BANZI; SHILOH, 2015).

Alguns *hardwares* são desenvolvidos para serem utilizados como módulos em placas específicas de arduino. A facilidade do ambiente de programação, a possibilidade de comunicação da placa do arduino com o computador, os módulos de hardwares, o preço acessível da placa do arduino são alguns dos motivos que tornam possíveis projetos com arduino serem de baixo custo, projetos multiplataforma e o surgir de extensões de software (SOLIMAN et al., 2013). Além disso, as placas do arduino podem ser montadas manualmente pelos projetistas, sem que seja necessário pagar por sua utilização (BANZI; SHILOH, 2015).

2.4 Perceptron Multicamadas (MLP)

O perceptron multicamadas (MLP) é um tipo de rede neural artificial de aprendizado supervisionado, que é principalmente utilizada nos casos em que a entrada é um conjunto de alta dimensão e discreto, e a saída é uma função real. O MLP é constituído por uma camada de entrada, uma camada de saída e uma ou mais camadas ocultas. Cada camada possui diversos nós, ou também chamados de neurônios, e são interconectadas entre si. Para cada conexão é atribuído um peso que é atualizado durante o processo de aprendizagem utilizando o conceito de *backpropagation*(FLORENCIO et al., 2018). Além disso, o MLP é capaz de reduzir o impacto negativo advindo de um possível dado com ruído que esteja presente no conjunto de entrada (MA et al., 2016). Geralmente, o número de camadas dessa rede neural e o número de neurônios em cada camada são definidos empiricamente (FLORENCIO et al., 2018).

2.5 Trabalhos Relacionados

PRABHU et al. realiza uma avaliação de sete técnicas para extração de assinatura única de memórias *flash*. As assinaturas são geradas através da utilização das mudanças randômicas do comportamento das memórias, essas mudanças de comportamento ocorrem por causa do processo de fabricação que indiretamente atribui unicidade aos ICs. As assinaturas podem ser detectadas através de uma interface elétrica comum, e as variações de fabricação podem ocorrer tanto entre *chips*, como entre os blocos/páginas que compõem o *array* de armazenamento da *flash*. Devido a isso, um único método de obtenção de assinaturas pode obter milhares de assinaturas diferentes de partes distintas de um único *chip*. Além disso, o trabalho propôs algoritmos para extração de assinaturas que durante execuções distintas, indica uma grande correlação entre assinaturas retiradas de um mesmo bloco de memória, e baixa correlação entre assinaturas retiradas de blocos distintos.

SUH; DEVADAS apresenta uma descrição do projeto para técnica PUF baseada na característica de *delay* dos fios e transistores, e descreve como um IC pode ser identificado através de uma autenticação de baixo custo (SUTAR; RAHA; RAGHUNATHAN, 2018). Além de especificar como operações criptográficas podem gerar Chaves Secretas Voláteis através desses ICs, e especificamente como cada assinatura pode ser considerada uma instância PUF. Os autores apresentam como uma forma de autenticação dos ICs pode ser utilizada baseada na geração de chave de criptografia obtida do hardware, e que pode ser executada através de um modelo de validação experimental da chave criptográfica.

WANG et al. demostraram que uma memória *flash* comercial e não modificada pode garantir duas importantes funções de segurança: a geração randômica de nú-

mero e a impressão digital do dispositivo. Tendo seu foco principal na geração de números aleatórios que são desenvolvidos por um algoritmo baseado nas características de *delay* de cada memória oriundas do processo de fabricação. Parcialmente a pesquisa avalia que taxas acima de 10Kbits / segundo permitem uma alta qualidade de obtenção de número aleatórios reais. Além disso, os autores basearam-se em uma programação de variações de limiar da tensão que permite uma rápida geração de muitas impressões únicas, que podem ser usadas para identificação e autenticação do dispositivo.

CAULFIELD et al. desenvolveram maneiras de utilizar a idiossincrasia para explorar características da memória *flash* de forma útil, como por exemplo, mensurar diretamente o desempenho e confiabilidade de um dispositivo. Para realizar uma análise detalhada sobre a memória *flash*, o autores adotaram as técnicas *Read Disturb e a Program Disturb* para simular disturbios, ou seja, aparecimento de erros nos dados da memória. Além disso, foi demonstrado e quantificado algumas características inesperadas desse tipo de memória e como a utilização de alguns passos pode aumentar o tempo de vida de um dispositivo *flash*.

SKOROBOGATOV em sua tese de doutorado, apresentou um extenso número de ataques à segurança de *hardware* em microcontroladores e *smartcards*. Detalhando conceitos chaves como os Ataques Invasivos, como a Engenharia Reversa, e Ataques Não-Invasivos, como o Ataque de *software*. Além disso, a tese também introduz um novo conceito de Ataques de *hardware* chamado Ataque Semi-Invasivo, o qual requer a retirada do invólucro do chip, mas não necessita de contato elétrico, ou seja, quase tão efetivo quanto um Ataque Invasivo mas pode ser baixo custo como os Ataques Não-Invasivos. Vale ressaltar que vários métodos e tecnologias de defesa e proteção são detalhadamente discutidos.

SKOROBOGATOV; ANDERSON descreve a nova classe de ataques em microcontroladores e *smartcards*, conhecida como Ataques Semi-Invasivos. Além disso, desenvolve técnicas para gerar falhas em uma Memória SRAM (Static Random Access Memory) através de uma fonte óptica. As falhas ocorrem quando um transistor conduz após ser exposto a determinado tipo de iluminação. Os autores apontam que qualquer bit pode ser setado ou resetado em um microcontolador. Vale ressaltar que a técnica utilizou uma lâmpada *flash* de uma câmera de 30 dólares para realizar os testes da técnica, mostrando que a técnica não necessita de laboratórios sofisticados e pode ser realizada em equipamentos de baixo custo.

RAHMATI et al. é uma pesquisa da área da Computação Aproximada que utiliza a variabilidade do tempo dos decaimentos de tensões em cada célula de memória DRAM como um erro padrão utilizado para criar uma impressão digital da memória. Para que isso fosse possível, foi desenvolvido uma distância métrica para comparar

impressões digitais da mesma memória ou memórias diferentes. Tal distância foi utilizada em um modelo matemático aproximado da DRAM para identificar as memórias com a utilização de um programa de manipulação de imagens.

A Tabela 2 apresenta um resumo sobre os trabalhos relacionados citados. Através da definição do tipo de memória utilizada na pesquisa, dos tipos de ataques a memórias abordados, das técnicas PUF utilizadas,dos níveis para proteção fornecidos e do *hardware* utilizado para extrair as assinaturas. Vale ressaltar que este trabalho de conclusão de curso utilizará uma técnica baseado na PUF *Program Operation Latency*, devido a suas vantagens quanto comparadas a outras técnicas. Este trabalho se diferencia dos demais trabalhos relacionados, que utilizaram segurança a nível de *hardware*, por utilizar a plataforma Arduino como ferramente de extração de assinatura. E além disso, por validar as assinaturas retiradas com a utilização do perceptron multicamadas, vide a Tabela 2.

Tabela 2 – Comparação dos Trabalhos Relacionados.

Trabalhos Relacionados	**Memória**	**Ataques**	**Técnica(s) PUF**	**Nível de Proteção**	**Extração**
(SKOROBOGATOV, 2005)	SRAM/ DRAM/ FLASH	Invasivos Não-Invasivos Semi-Invasivos		*Hardware*	
(SKOROBOGATOV; ANDERSON, 2002)	SRAM	Semi-Invasivos		*Hardware*	
(TEHRANIPOOR et al., 2015)	DRAM	Invasivos		*Hardware*	
(RAHMATI et al., 2016)	DRAM	Não-Invasivos		*Hardware/ Software*	FPGA
(WANG et al., 2012)	*Flash*	Invasivos	Program Disturb	*Hardware*	FPGA
(CAULFIELD et al., 2009)	*Flash*	Invasivos	*Program Disturb, Read Disturb.*	*Hardware*	FPGA
(PRABHU et al., 2011)	*Flash*	Invasivos	*Program Disturb, Read Disturb, Program Operation Latency.*	*Hardware*	FPGA
Trabalho de Conclusão de Curso	*Flash*	Invasivos	*Program Operation Latency*	*Hardware*	Arduino

3 Metodologia

Este capítulo apresenta os procedimentos metodológicos adotados nesta pesquisa. Inicialmente, é apresentada uma visão geral da metodologia e, em seguida, são apresentados os procedimentos, materiais utilizados para configuração e implementação da solução.

3.1 Visão Geral

Esta seção apresenta a sequência de atividades e materiais que foram utilizados para execução do algoritmo de identificação de memórias *flash* baseado em PUF. A Figura 1 ilustra a metodologia adotada e contextualiza o ambiente no qual este trabalho está inserido, destacamos os seguintes passos: configuração do ambiente, extração de assinaturas de identificação, armazenamento em arquivos, formatação de arquivos, utilização da plataforma Weka, perceptron multicamadas e avaliação de blocos representativos da memória *Flash*. Além disso, foram disponibilizados os pseudo-códigos que devem ser utilizados por um Arduino e por um computador para extração, transferência e armazenamento dos dados. Os quais são obtidos pela execução do algoritmo no Arduino, e posteriormente são enviados para o computador via entrada serial.

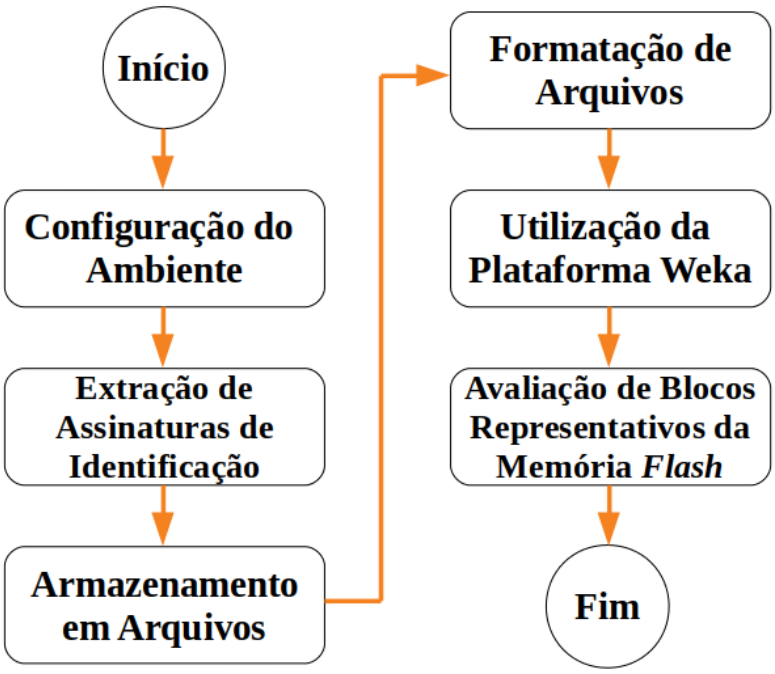

Figura 1 – Visão Geral da Metodologia Proposta.

3.1.1 Configuração do Ambiente

Para realizar a prototipação do ambiente do projeto foram utilizados os seguintes materiais: 1) O Arduino Mega 2560 (ver Figura 2), que foi utilizado devido a facilidade de prototipação. A plataforma Arduino fornece a possibilidade de programação na linguagem C++, apresenta facilidade na utilização dos pinos de entradas e saídas, entre outras. A plataforma Arduino Mega é geralmente utilizado em projetos que exijam maior desempenho quando comparado a outras do mesmo fabricante. 2) Um módulo adaptador de cartão microSD (ver Figura 3), que foi utilizado por sua compatibilidade com a plataforma Arduino, auxiliando ainda na redução de tempo da prototipação do projeto. 3) Um Cartão MicroSD de 2GB (ver Figura 4), que foi utilizado por ser compatível com o módulo adaptador de cartão microSD e com a biblioteca padrão FAT.h. Essa biblioteca está presente no ambiente de desenvolvimento integrado da plataforma Arduino.

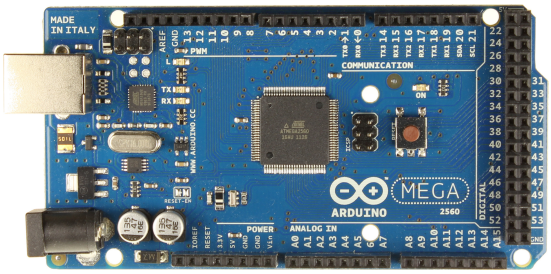

Figura 2 – Arduino Mega 2560.
Fonte: (ELECTRO SCHEMATICS, 2014).

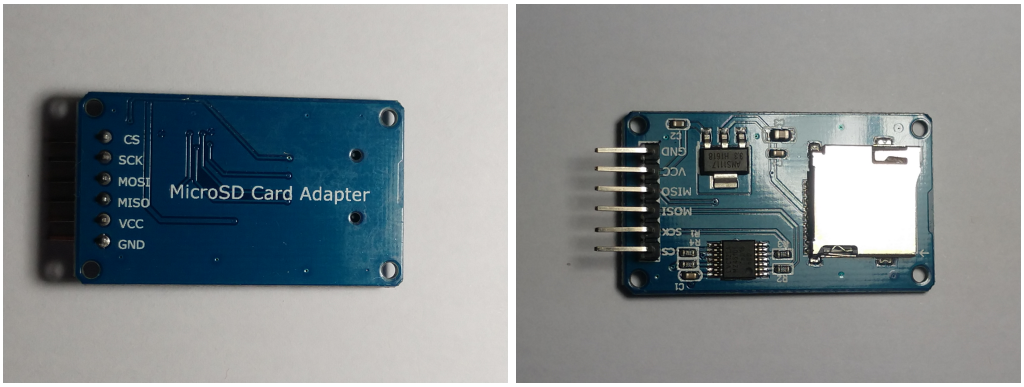

Figura 3 – Módulo Adaptador de Cartão MicroSD.

No processo de montagem do protótipo, consideramos as especificações para utilização do módulo adaptador de cartão microSD no Arduino Mega. Esse módulo possui dois pinos de alimentação, que podem ser conectados diretamente ao VCC[1] e GND[2] presentes na pinagem do Arduino, e outros quatro pinos dedicados ao padrão de comunicação chamado SPI. Esses quatro pinos são geralmente denominados como

[1] Voltagem em corrente contínua, ou nível alto.
[2] Do inglês, *graduated neutral density filter*, que significa filtro de densidade neutra, ou terra.

Figura 4 – Cartão MicroSD.

MOSI, MISO, SCK, CS e tem correspondência a pinos do Arduino, que podem variar dependendo do modelo. Particularmente, no caso do Arduino Mega 2560 o MOSI, MISO, SCK e CS são localizados nos pinos 51, 50, 52, 53 respectivamente. Após realizar as conexões entre as portas e pinos correspondentes, é necessário inserir o cartão SD no módulo adaptador na posição correta. A Figura 5 apresenta um exemplo do módulo de cartão SD conectado ao Arduino Mega 2560.

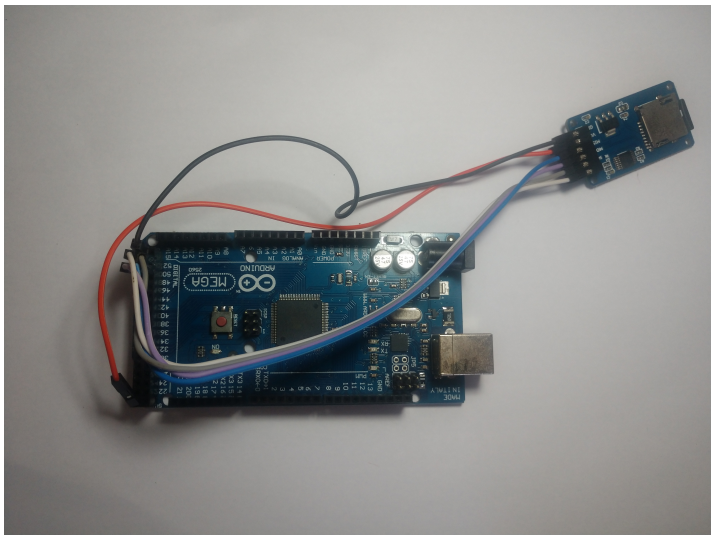

Figura 5 – Exemplo de um circuito após montagem dos componentes.

3.1.2 Extração de Assinaturas de Identificação

Após a montagem do circuito, faz-se necessário conectar a placa do Arduino por meio de uma comunicação serial USB a um computador. Essa conexão tem o objetivo de transferir os dados coletados visando a extração de assinaturas de identificação. Para realizar essa atividade, foi adotado e implementado o seguinte Algoritmo (1). É importante mencionar sobre a necessidade da utilização da biblioteca Serial.h para que a comunicação ocorra, visto que, esta é responsável pelo envio dos dados via USB. Além disso, o computador deve utilizar o *software Processing* para fazer a leitura dos dados na entrada serial do computador. As assinaturas de identificação da memória *flash* são obtidas pelo Arduino através da utilização de bibliotecas para utilização do cartão microSD, podemos citar as bibliotecas SD.h e Sd2Card.h, que vem no *software* padrão do arduino. Depois de extraídas, as assinaturas de identificação da memória *flash* são transferidas e armazenadas no disco do computador conectado. Vale ressaltar que as assinaturas de identificação são obtidas pela execução do algoritmo 1 no Arduino Mega 2569. O pseudocódigo do algoritmo de identificação de memórias *flash*, baseia-se na técnica PUF.

Algorithm 1: Pseudo-código de Extração de Assinaturas de Identificação de uma Memória *Flash* no Arduino.

Variáveis de Entrada:
$blocksAddress[\ ]$
Variável de Saída:
$times[blocksLength][lastCycle]$ $\quad //Enviado\ via\ Serial\ para\ o\ computador$
Variáveis Auxiliares:
$blocksLength$ **=** $blocksAddress.lentgh$
Início:
  Para $cycle$ **De** 1 **Até** $lastCycle$ **Faça:**
    Para b **De** 1 **Até** $blocksLength$ **Faça:**
      $times[cycle][b] \leftarrow timeOperation(blocksAddress[b])$
    Fim
  Fim
Fim

3.1.3 Armazenamento em Arquivo

Nesta fase, os dados coletados precisam ser armazenados no disco rígido do computador conectado via USB. O Algoritmo 2 descreve o pseudocódigo adotado para realizar esta etapa. O algoritmo é capaz de ler a entrada USB e salvar as assinaturas de identificação das memórias *flash*, enviadas pelo Arduino, em um arquivo. O pseudocódigo do algoritmo utilizado na ferramenta *Processing* é responsável pela leitura da USB.

Algorithm 2: Pseudo-código de Armazenamento das Assinaturas de Identificação de uma Memória *Flash* no Computador.

```
1  : Variáveis Auxiliares:
2  : assinaturaIdentificacao
3  : file
4  : Início:
5  :     waitArduino()     //Tempo para o arduino comear a escrever na Serial
6  :     file ← newfile()
7  :     file ← headWeka()   //Inicio do arquivos arff
8  :     Enquanto Serial.available() Faça:
9  :         assinaturaIdentificacao ← Serial.read()
10 :         file.addLine(formatoWeka(assinaturaIdentificacao))
11 :         waitArduino()     //Tempo para o arduino comear a escrever na Serial
12 :     Fim
13 :     file.save()
14 :     file.close()
15 : Fim
```

Vale ressaltar que o computador é consideravelmente mais rápido que o Arduino, logo se faz necessário a utilização de *delays* durante a execução do código no Processing, para que o Arduino tenha tempo hábil de enviar as assinaturas via serial. Nesse sentido, adotamos a função *waitArduino()* no Algoritmo 2. Contudo, esse valor depende exclusivamente da interação dos tempos relacionados aos dispositivos utilizados (Arduino, Módulo, Computador), devido a isso foi atribuído um valor empírico oriundos dos testes realizados após montagem do circuito da Figura 5.

3.1.4 Formatação de Arquivos

Após a coleta e o armazenamento dos dados no disco do computador conectado ao circuito, faz-se necessário verificar se os dados foram coletados corretamente. A Figura 6 apresenta um exemplo da estrutura do arquivo gerado. O arquivo gerado pelo Algoritmo 2 apresenta várias extrações de uma mesma sequência de blocos de uma memória *flash*, ou seja, do cartão microSD. Cada linha do arquivo representa uma assinatura de identificação, essa linha possui uma sequência de números, separados por espaço, na mesma quantidade dos blocos. Os valores representam o tempo de operação de um bloco de memória.

```
2152 2320 2312 2312 2312
2312 2312 2312 2160 2160
2160 2160 2168 2276 2276
2272 2272 2272 2272 2272
        ...
2272 2312 2316 2320 2320
2328 2320 2312 2312 2312
2312 2312 2312 2160 2160
2160 2160 2168 2276 2276
```

Figura 6 – Exemplo do arquivo para uma sequência de cinco blocos.

A utilização do arquivo da Figura 6 depende exclusivamente da escolha do *software* que será utilizado para validação das assinaturas de identificação da memória *flash*. Cada *software* possui uma forma específica de formatação para os arquivos de entrada, essa formatação deve ser respeitada para que a execução do *software* ocorra com sucesso. Neste trabalho foi utilizado o Perceptron Multicamadas, para verificação dessas assinaturas de identificação. A Figura 7 representa o esquema do fluxo da interação entre o Arduino, o computador e o Perceptron Multicamadas.

Figura 7 – Esquema básico do fluxo de interação.

3.1.5 Utilização da Plataforma Weka

Esta fase, tem o intuito de realizar uma verificação das assinaturas de identificação de memória *flash*. Assim, a partir dos dados armazenados utilizamos o conceito do *Perceptron Multicamadas*. Para realizar esta etapa, selecionamos a ferramenta Weka (WITTEN et al., 2016). O Weka é um *software* livre que contém diversas técnicas de mineração de dados de forma gratuita (Ver Figura 8), por exemplo, o Perceptron Multicamadas. Contudo, o arquivo com os dados coletados precisou ser convertido para o formato da ferramenta Weka, o `".arff"`.

Figura 8 – Tela inicial do Weka.

3.1.6 Avaliação dos Blocos Representativos da Memória *Flash*

A principal razão para a utilização do perceptron multicamadas deve-se a visualização dos dados obtidos nas coletas das assinaturas de identificação, vide a Figura 6. Pois, esses dados podem não ser linearmente separáveis. Motivo pelo qual é necessário se verificar as assinaturas de identificação de uma memória *flash* através da utilização do perceptron multicamadas (LI et al., 2019). Este será treinado com os tempos de operação obtidos no Algoritmo 1 com o intuito de que seja bem treinado e que aprenda o suficiente dos dados passados para generalizar no futuro, já que as assinaturas de identificação podem possuir variações de tempo de operação em um mesmo bloco.

A camada de entrada do Perceptron Multicamadas será estimulada pelos valores de operação da sequência de blocos (conforme exemplo da Figura 6). Os valores relacionados ao bloco n estimulam o nó n da camada de entrada. Todo esse processo é realizado pela execução do *software* Weka configurado para o Perceptron Multicamadas. Entretanto, para que isso seja possível, o arquivo precisa seguir a formatação padrão `".arff"`.

A validação cruzada é uma técnica para avaliar o poder de generalização de um classificador, que pode ser utilizada através da ferramente Weka, onde o conjunto de dados de entrada disponível é particionado aleatoriamente em um conjunto de treinamento e um conjunto de teste. Com o objetivo de validar o modelo em um conjunto de dados diferente do usado para a estimativa de parâmetros. Além disso, o Weka fornece os pesos resultantes de cada nó da camada de entrada do Perceptron Multicamadas.

3.1.7 Métricas Definidas Por Padrão no Weka

Por fim, vale especificar as métricas utilizadas pela plataforma Weka, e que foram apresentadas após os treinamentos do MLP. As métricas são: 1) Amostras corretamente classificadas (do inglês, *Correctly Classified Instances*); 2) Amostras incorretamente classificadas (do inglês, *Incorrectly Classified Instances*); 3) Coeficiente de *Kappa* (do inglês, *Kappa statistic*) tem a finalidade de medir o grau de concordância entre proporções derivadas de amostras dependentes. O *Kappa* possibilita a avaliação tanto se a concordância está além do esperado tão somente pelo acaso, quanto pelo grau dessa concordância. Essa métrica tem como valor máximo o valor unitário, que representa total concordância, e valor igual a 0 que indica concordância nula (GUIMARÃES, 2017). 4) O Erro Absoluto médio (do inglês, *Mean absolute error*), caracteriza-se por ser a média dos erros cometidos pelo modelo de previsão durante uma série de execuções. Indicando a média do afastamento de todos os valores fornecidos pelos classificadores e o seu real valor (MARTINS, 2016). 5) Raiz quadrada do erro quadrático médio (do inglês, *Root mean squared error*), é comumente utilizado nos modelos preditivos, sendo útil na medida em que o erro avaliado é da mesma magnitude que a quantidade a ser prevista (NUNES, 2017). 6) Erro relativo absoluto (do inglês, *Relative absolute error*). 7) Raiz do erro relativo absoluto (do inglês, *Root relative squared error*). 8) Número total de instâncias (do inglês, Total Number of Instances) presentes no arquivo `".arff"`.

4 Resultados Experimentais

Este capítulo apresenta os resultados obtidos durante a execução da metodologia proposta, além das lições aprendidas com os experimentos realizados. Além disso, para realizar os experimentos foi necessário definir um plano experimental, o qual foi descrito abaixo.

4.1 Plano de Experimento

Para realizar o conjunto de passos proposto na metodologia (Ver Figura 7), foi necessário especificar alguns parâmetros dos Algoritmos 1 e 2. Inicialmente, foi definido para esta pesquisa que o tempo de operação do Algoritmo 1 representa a operação de leitura da memória *flash*. Vale relembrar que se trata de um cartão microSD, como definido na Seção 3.1.1. Contudo, a operação de leitura do cartão microSD pode dar origem ao surgimento de erros na execução dessa operação, devido às características intrínsecas da fabricação e especificação do cartão microSD.

Um erro comum identificado na realização da operação de leitura, tinha como causa, a tentativa de acessar algum endereço inexistente, ou quando o cartão de memória estava ocupado com outra operação. Isso resultou na captura de tempos de operação muito próximos ou iguais a zero. Nesse sentido, visando evitar tais situações que poderiam influenciar na captura real da assinatura de identificação da memória *flash* foi necessário remover tais valores. Assim, quando qualquer operação de leitura retornava valores muito próximos a zero, o que indicava a presença de um erro, foi realizado um outra operação de leitura na mesma região visando capturar o tempo de operação válido. Outro ponto importante, foi a definição da grandeza de tempo utilizada para aferir o tempo da operação de leitura. Para solucionar isso, foi realizado um teste empírico com o tempo de operação de leitura no modelo da Figura 5, e os resultados foram capturados quando utilizamos a grandeza em microssegundos. Devido a isso, foi utilizado a função `Micros()`, sendo esta disponibilizada na biblioteca padrão do Arduino.

O *array* que representa a assinatura de identificação da memória *flash* foi definido considerando todos os tempos da operação de leitura em um conjunto de blocos. Onde, o tempo de leitura aferido no enésimo bloco foi armazenado na enésima posição desse *array*. Especificamente, o conjunto de blocos utilizado foi uma sequência de blocos simples, ou seja, o bloco `n` é fisicamente adjacente ao bloco `(n+1)`. Além disso, outra definição foi devido a limitação de caracteres que podem ser escritos em cada linha de um arquivo `".arff"`. Pois, cada assinatura de identificação fica em uma linha

por padrão, por isso foi definido que seria utilizado um limite de blocos por assinatura. Após a realização de testes de escrita no arquivo, definimos um limite de 40 (quarenta) blocos, ou seja, esse foi o valor utilizado na variável `blocksLength` do Algoritmo 1.

O parâmetro `lastCycle`, presente no Algoritmo 1, que representa o número de amostras coletadas pelo Arduino foi definido com um valor que não implicasse em uma alta taxa de espera de execução do Arduino, o que impediria a utilização da técnica proposta em vários cenários. Além disso, como esse parâmetro influencia diretamente no treinamento do Perceptron Multicamadas, foram realizados diversos testes com variados tamanhos de arquivos, ou seja, com diversas quantidade de assinaturas de identificação coletadas. Geralmente a quantidade de amostras necessárias para o treinamento do MLP é definido de forma empírica, pois depende de diversos fatores como, por exemplo, a repetitividade dos dados. Por isso, foram realizados testes com 50, 100, 200, 400, 800 e 16000 amostras por arquivo. Nesse sentido, foram realizados testes e o valor de 1000 (mil) amostras demonstrou ser suficiente para o treinamento do Perceptron Multicamadas.

4.2 Memória *Flash*

Para realizar a extração das assinaturas de identificação utilizando o Arduino no Cartão MicroSD, com base no Algoritmo 1, foi necessário também definir os endereços dos blocos da memória. Foram utilizadas as bibliotecas `SD.h` e `Sd2Card.h` (veja a Seção 3.1.2), assim como, consideramos os endereços na notação hexadecimal. Além disso, utilizamos um vetor com 40 (quarenta) endereços sequências, por exemplo, o endereço `0x000F0EEE`. Para inicializar o parâmetro de endereço, utilizamos a variável de entrada `blocksAddress[ ]` no Algoritmo 1.

Outro parâmetro que foi definido diz respeito ao tipo de operação que o Cartão MicroSD deve realizar para então ser observado. Contudo, a extração das assinaturas de identificação não deve possuir um tempo demasiadamente custoso, pois assim, a utilização dessa metodologia seria inviável para os testes ou até para aplicações futuras. Devido a isso, as operações primárias da memória são mais recomendadas, como por exemplo, a operação de leitura de um bloco na memória.

4.3 Exemplo de Arquivos

O arquivo gerado pelo Algoritmo 2, já contempla o formato `".arff"`, sendo representado pela junção de um cabeçalho e os exemplos dos dados. O cabeçalho apresenta os parâmetros que são utilizados pela ferramenta Weka, por exemplo, o tamanho da assinatura de identificação, definida pela quantidade de blocos. Estes, por sua vez,

definem a quantidade de atributos no cabeçalho, onde cada bloco corresponde ao atributo. Nesse caso, a assinatura de identificação é numérica, pois representa o tempo da operação de leitura no bloco, logo cada atributo é definido como numérico. A Figura 9 apresenta um exemplo de cabeçalho obtido no experimento.

```
@relation teste
@attribute block0 numeric
@attribute block1 numeric
@attribute block2 numeric
@attribute block3 numeric
@attribute block4 numeric
@attribute block5 numeric
        ...
@attribute block35 numeric
@attribute block36 numeric
@attribute block37 numeric
@attribute block38 numeric
@attribute block39 numeric
@attribute node {'yes','no'}
```

Figura 9 – Exemplo do cabeçalho do arquivo `".arff"`.

Outra parte do arquivo `".arff"` é o exemplo dos dados, neste caso específico cada linha representa uma assinatura de identificação da memória *flash*. Essas assinaturas são definidas como uma sequência de números separados por virgula. Vale ressaltar que, o Algoritmo 2 possui uma função `formatoWeka()` que é responsável por definir o formato de cada linha do arquivo. Podemos citar como exemplo o último atributo da Figura 9, onde temos mais de um parâmetro. Assim, a partir desse parâmetro, a ferramenta Weka poderá verificar se a assinatura de identificação de determinada linha é de uma mesma sequência de blocos ou de outra. A Figura 10 apresenta um exemplo com um conjunto de tempos coletados.

```
@data

2324,2312,2316,2308,2316,2312, ... ,2276,2284,2284,2280,2280,'yes'
2316,2316,2316,2320,2324,2320, ... ,2320,2320,2316,2316,2316,'yes'
2316,2320,2320,2320,2328,2320, ... ,2316,2316,2316,2316,2316,'yes'
2316,2316,2276,2276,2276,2284, ... ,2284,2284,2280,2284,2276,'yes'
2316,2316,2276,2284,2284,2280, ... ,2284,2284,2276,2276,2276,'yes'
2316,2320,2284,2280,2284,2284, ... ,2276,2276,2276,2276,2276,'yes'
                               ...
2320,2320,2292,2284,2280,2276, ... ,2276,2276,2276,2276,2276,'no'
2320,2320,2284,2276,2276,2276, ... ,2276,2276,2276,2276,2276,'no'
2320,2312,2276,2276,2276,2276, ... ,2276,2276,2276,2280,2284,'no'
2316,2316,2276,2276,2276,2276, ... ,2276,2284,2284,2280,2280,'no'
2316,2316,2276,2276,2276,2284, ... ,2284,2280,2280,2280,2276,'no'
2316,2312,2276,2276,2284,2280, ... ,2280,2284,2276,2276,2276,'no'
2316,2320,2284,2280,2288,2284, ... ,2280,2276,2276,2276,2276,'no'
                               ...
```

Figura 10 – Exemplo de linhas que representam as assinaturas de identificação de uma memória *flash* no formato de arquivo `".arff"` do Weka.

O atributo `node` da Figura 9 indica para o Weka que cada linha pode ser classificada como `yes/no` (sequência distintas de blocos). Esse atributo é utilizado para a execução do Perceptron Multicamadas, o qual necessita ser estimulado diversas vezes com um conjunto de assinaturas de identificação. Esse conjunto pode ser particionado automaticamente pela ferramenta Weka com o intuito de gerar um conjunto de treinamento, utilizado para o aprendizado supervisionado, e outro conjunto de teste. Esses estímulos são utilizados pelo Perceptron Multicamadas para calcular e atualizar a estrutura interna da rede neural, atualizando os pesos em um processo conhecido como *backpropagation*. O objetivo desse procedimento de aprendizado por sucessivas correções é ajustar os parâmetros da rede para que a resposta visualizada se aproxime da resposta almejada.

O conjunto de assinaturas que foram utilizado para o experimento apresentou as seguintes características: 1) As assinaturas de `yes/no` são de regiões distintas de uma mesma memória, ou mesma região e memórias distintas; 2) 40% (quarenta) por cento do conjunto de assinaturas de identificação foram utilizadas como conjunto de

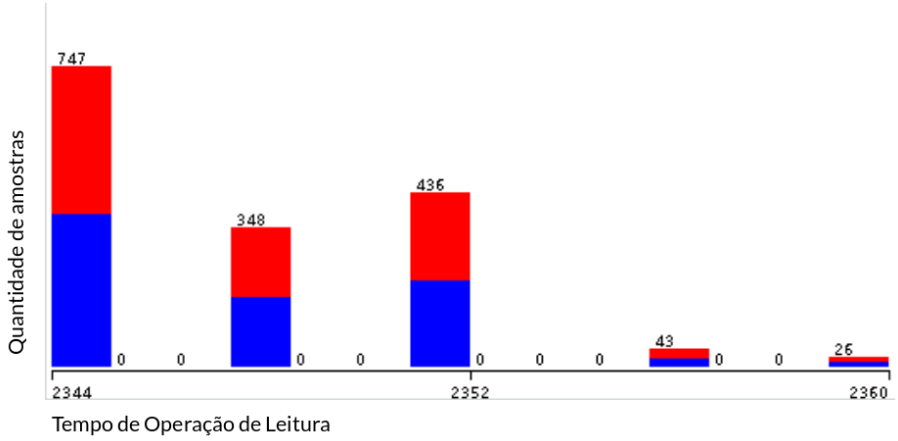

Figura 11 – Exemplo de bloco com tempos de operação de leitura que não pode ser separado linearmente.

treinamento; 3) A quantidade de assinaturas não ultrapassou mais de 2000 (duas mil) assinaturas.

4.4 Avaliação dos Blocos Representativos

O arquivo de entrada utilizado na plataforma Weka, que serviu como base para execução do perceptron multicamadas, possui uma quantidade de 1600 (mil e seiscentas) assinaturas de identificação da memória. Essas assinaturas foram coletadas através da execução dos Algoritmos 1 e 2 em dois endereços diferentes de um mesmo cartão microSD ou, em alguns casos, os mesmos endereços de memória em cartões microSD distintos. O Weka fornece um gráfico cartesiano de distribuição dos dados após o carregamento do arquivo na plataforma. A partir do gráfico disponibilizado, foi possível notar que os tempos de operação de leitura de um mesmo bloco, ou seja, a mesma posição do *array* em todas as assinaturas coletadas, podem ser representados de forma geral em dois grandes grupos de mesmo tamanho (Devido ao padrão *node* adotado no arquivo, vide a Seção 4.3).

A Figura 11 apresenta a distribuição cartesiana dos 1600 (mil e seiscentos) tempos de operação de leitura em um único bloco, ou seja, de um mesmo atributo. Esses tempos foram representados em um gráfico de barras, com cores distintas para cada grupo. Contudo, as barras podem inclusive ser apresentadas sobrepostas para indicar a proporção entre a aferição daquele tempo específico, em cada grupo, em determinado valor ou intervalo. Particularmente, na Figura 11 os tempos apresentados não podem ser separados linearmente. Por outro lado, há outros blocos que apresentam outros tipos de distribuições, por exemplo o demonstrado na Figura 12. Na Figura 12, a distribuição apresentada indicou tempos de operação de leitura onde não há sobreposição entre as classes, ou seja, onde os tempos podem ser linearmente separáveis.

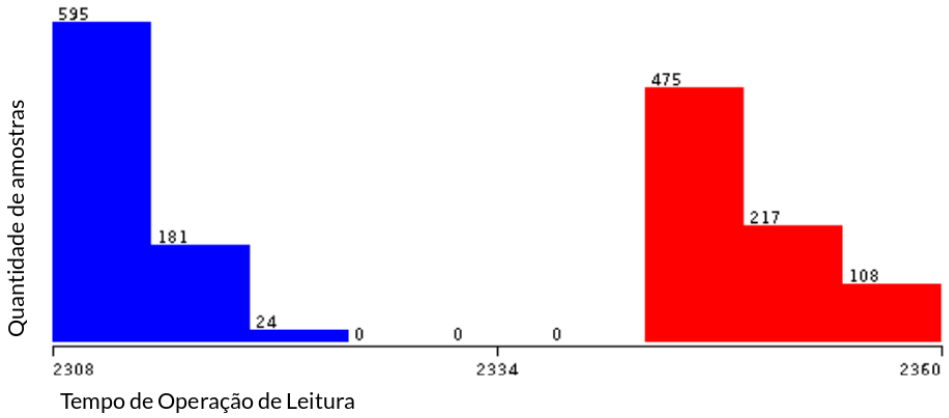

Figura 12 – Exemplo de bloco com tempos de operação de leitura linearmente separáveis.

Devido a essas variações de tempos entre cada bloco da assinatura foi sugerido a utilização do perceptron multicamadas. Para que fosse bem treinado e que aprendesse o suficiente sobre as assinaturas de identificação para generalizar no futuro, ou seja, tornando-o possível identificar se tal assinatura é ou não de uma região específica da memória *flash*. O que levou a identificar a própria memória.

4.4.1 Resultados

Após a configuração do Weka, o perceptron multicamadas recebeu uma base de dados contida em um arquivo `".arff"` como, por exemplo, com 1600 (mil e seiscentas) assinaturas de identificação. Visando identificar um padrão nesse grupo que a princípio poderia não ser linearmente separável. Esse grupo de assinaturas de identificação foi oriundo da junção de um conjunto de assinaturas extraídas de uma sequência de blocos específicos de um cartão microSD, com outro conjunto com a mesma quantidade de assinaturas de identificação extraídas de outra sequência de blocos. Foram realizados testes onde esse outro conjunto foi obtido tanto do mesmo cartão, mas em posições distintas da memória, quanto de um cartão microSD diferente, mas com a mesma posição da memória.

O Weka utiliza cada assinatura de identificação presente no arquivo `".arff"` durante o treinamento do perceptron multicamadas. Cada posição do *array* das assinaturas de identificação foi associada a um nó da camada de entrada do perceptron multicamadas. Dessa forma, a cada leitura de uma nova assinatura de identificação o perceptron multicamadas atualizava os seus pesos referentes as conexões entre as camadas, no processo de aprendizado baseado no conceito de *backpropagation*, como foi dito na seção 2.4. Após o treinamento com a base de dados citada, utilizamos o Weka para realizar uma validação cruzada de 10 partes, a qual foi disponibilizada pela própria ferramenta, vide a Figura 13.

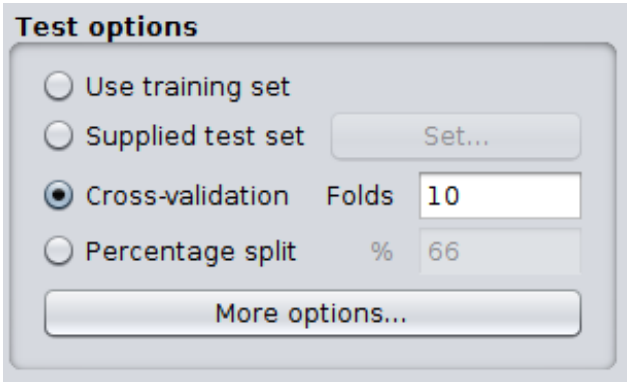

Figura 13 – Configuração da plataforma Weka para validação cruzada de 10 partes.

Essa avaliação cruzada de 10 partes obteve alguns resultados que demonstram que perceptron multicamadas foi capaz de aprender o suficiente sobre as assinaturas de identificação para generalizar no futuro, ou seja, para que fosse possível classificar o grupo a que cada assinatura pertence. A Figura 14 apresenta o sumário dos dados finais obtidos pela plataforma Weka após a execução de todo o processo com um banco de dados de assinaturas de identificação oriundos do mesmo cartão microSD mas com regiões distintas. Nesse resultado, vale destacar que a matriz de confusão encontrada demonstra que o *perceptron* foi capaz de classificar de forma correta todas

as assinaturas encontradas, e o valor do erro absoluto foi de $0,1325\%$. Podemos citar também a estatística *Kappa* que apresentou o valor 1, sendo assim, um alto nível de concordância.

```
Time taken to build model: 7.63 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1600               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.0007
Root mean squared error                  0.0007
Relative absolute error                  0.1325 %
Root relative squared error              0.1365 %
Total Number of Instances             1600

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     yes
               1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     no
Weighted Avg.  1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000

=== Confusion Matrix ===

   a   b   <-- classified as
 800   0 |   a = yes
   0 800 |   b = no
```

Figura 14 – Status da plataforma Weka após validação cruzada de 10 partes, com 1600 assinaturas de identificação divididas igualmente por classe.

A Figura 15 apresenta o sumário dos resultados obtidos pela execução do *perceptron* multicamadas na plataforma Weka após a execução de todo o processo com um banco de dados de assinaturas de identificação oriundos de cartões microSD distintos. A matriz de confusão também foi capaz de classificar de forma correta todas as assinaturas do arquivo, e o valor do erro absoluto obtido foi de $0,1822\%$. A estatística *Kappa* neste resultado também apresentou o valor 1, ou seja, demonstrando um alto nível de concordância.

A quantidade de assinaturas de identificação que foi utilizada no arquivo `".arff"` não pôde ser demasiadamente pequena, devido principalmente a possibilidade de perda da representatividade dos dados. A Figura 16 mostra um exemplo dessa perda. As nuvens representam os tempos das coletas de assinaturas do mesmo bloco em memórias diferentes, respectivamente representada pela cor azul e vermelha. Dessa forma, é possível observar que diversas extrações da classe no (representada pela cor vermelha) apresentaram valores de tempos aproximados com a classe *yes* (representada pela cor azul). Nesse sentido, caso utilizássemos um conjunto reduzido de assinaturas de identificação, o perceptron multicamadas poderia não ser capaz de generalizar corretamente.

```
Time taken to build model: 3.85 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         800               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.0009
Root mean squared error                  0.0009
Relative absolute error                  0.1822 %
Root relative squared error              0.1892 %
Total Number of Instances              800

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     yes
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     no
Weighted Avg.    1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000

=== Confusion Matrix ===

   a   b   <-- classified as
 400   0 |   a = yes
   0 400 |   b = no
```

Figura 15 – Status da plataforma Weka após validação cruzada de 10 partes, com 400 assinaturas de identificação por classe.

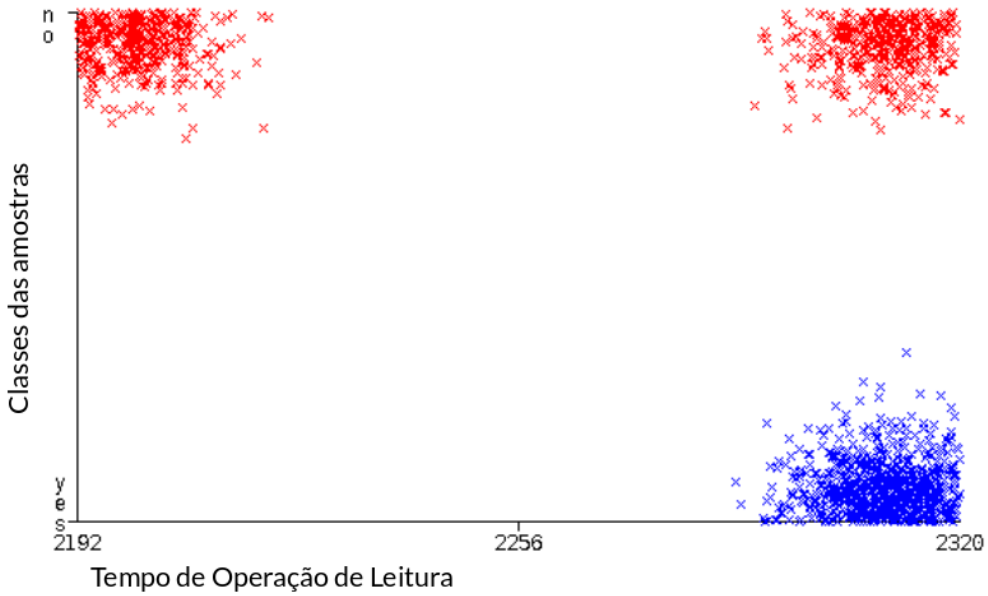

Figura 16 – Distribuição dos tempos coletados no décimo segundo bloco da assinatura de identificação em memórias diferentes, com endereço físico de $0x000F0EF9$.

Outro ponto importante foi a necessidade de um bom treinamento para o perceptron multicamadas, pois quanto maior a quantidade de exemplos de assinaturas maior a atualização dos pesos do MLP, ou seja, uma maior possibilidade de aperfeiçoamento. Contudo, nos testes em que o arquivo `".arff"` possuiu um grande número de exemplos, o tempo de execução do treinamento foi largamente excessivo, o que

tornaria a reprodução praticamente inviável e levaria a necessidade de se reduzir o tamanho do arquivo. Desta forma, com o intuito de demonstrar uma experimentação com uma quantidade reduzida de amostras, realizamos uma análise com 85 amostras. A Figura 17 apresenta o sumário estatístico.

```
Time taken to build model: 0.42 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          74               87.0588 %
Incorrectly Classified Instances        11               12.9412 %
Kappa statistic                          0.6346
Mean absolute error                      0.1635
Root mean squared error                  0.3447
Relative absolute error                 43.5346 %
Root relative squared error             79.8625 %
Total Number of Instances               85

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,667    0,063    0,778      0,667    0,718      0,638    0,807     0,769     yes
                 0,938    0,333    0,896      0,938    0,916      0,638    0,807     0,911     no
Weighted Avg.    0,871    0,266    0,866      0,871    0,867      0,638    0,807     0,876

=== Confusion Matrix ===

  a  b   <-- classified as
 14  7 |  a = yes
  4 60 |  b = no
```

Figura 17 – Status da plataforma Weka após validação cruzada de 10 partes, com 74 assinaturas de identificação.

Após analisar as métricas do sumário estatístico da plataforma Weka, verificamos a partir das Figuras 14 e 15 que as assinaturas de identificação foram corretamente classificadas. Além disso, o coeficiente de *Kappa* obtido indicou total concordância. O erro absoluto médio, a raiz quadrada do erro quadrático médio, o erro relativo absoluto, e a raiz do erro relativo absoluto apresentaram valores muito próximos a zero o que demonstra qualidade na classificação.

Já no experimento ilustrado na Figura 17, apenas $87,05\%$ das amostras foram corretamente classificadas. Além disso, o coeficiente de *Kappa* obtido obteve o valor 0.63, indicando uma baixa concordância. E o erro absoluto médio, a raiz quadrada do erro quadrático médio, o erro relativo absoluto apresentaram aumento significativos em suas porcentagens. Demonstrando assim a importância de uma quantidade significativa de amostras para o treinamento do MLP, que utilizou os parâmetros padrões que foram fornecidos pela própria plataforma, sem alteração do autor.

5 Conclusões e Trabalhos Futuros

Nesse trabalho foram feitas extrações de assinaturas de identificação de memória *flash*, através de uma metodologia baseada em técnicas PUF com a utilização de um Arduino mega. Além disso, foi utilizada a plataforma Weka com um arquivo `".arff"` tendo o intuito de treinar um perceptron multicamadas. O qual foi capaz de aprender o suficiente sobre essas assinaturas de identificação e generalizar no futuro, sendo capaz de classificar corretamente as classes das assinaturas de identificação da memória *flash*. O comportamento da operação de leitura de uma memória *flash* em um cartão microSD é capaz de fornecer uma assinatura de identificação, o que pode ser utilizada para o diferenciar dos demais. Como os *chips flash* são largamente utilizados, a metodologia proposta tem o potencial de ser amplamente aplicada em diversos dispositivos eletrônicos atuais, devido principalmente a utilização de componentes eletrônicos de baixo custo.

5.1 Contribuições

Assim sendo, as contribuições deste trabalho de graduação estão descritas abaixo:

- **Metodologia:** A metodologia aborda uma estratégia que utiliza uma sequência de etapas para a extração de assinaturas de identificação de um cartão microSD através da utilização do Arduino mega. Além disso, fornece os passos para validação de assinaturas de identificação de memória *flash* com a utilização de um *perceptron* multicamadas.
- **Circuito:** O circuito apresentado teve o intuito de diminuir o tempo de prototipação. Por causa da modularidade de seus componentes, e pela facilidade da criação de réplicas. Além disso, apresenta um baixo custo de aquisição dos componentes eletrônicos, quando comparado com a plataforma *FPGA* (do inglês, *Field Programmable Gate Array*, em português ”Arranjo de Portas Programáveis em Campo”) que é utilizada na maioria dos trabalhos mencionados. Outro ponto importante é que os componentes desse circuito são acessíveis em plataformas populares de *e-commerce*.
- **Algoritmo de Extração:** Esse trabalho apresentou o algoritmo de extração de assinaturas de identificação com a utilização da plataforma Arduino, o que auxilia aos leitores a rápida reprodução dos testes aqui descritos. Além disso, foram

informados os passos de configuração da ferramenta Weka para o treinamento do perceptron multicamadas e validação das assinaturas de identificação.

5.2 Limitações

As limitações relacionadas ao desenvolvimento deste trabalho de conclusão de curso está relacionada principalmente a interação dos seu componentes. Podemos descrever os seguintes pontos: 1) O processo de extração de assinatura com a utilização do Arduino Mega é limitado, devido a capacidade de processamento do próprio Arduino, da velocidade de iteração com o cartão microSD, e da capacidade de envio dos dados via serial para o computador; 2) A capacidade de processamento do computador é importante, devido a utilização da ferramenta Weka na etapa de treinamento do perceptron multicamadas. Esta etapa pode apresentar um tempo variado de acordo com a configuração do computador utilizado; 3) Outra limitação está relacionada com a representatividade dos dados. A quantidade de assinaturas de identificação fornecidas ao perceptron multicamadas deve apresentar uma variedade considerável, e definida empiricamente, para que o MLP possa ser bem treinado e assim saber generalizar no futuro.

5.3 Trabalhos Futuros

Alguns pontos podem ser considerados como trabalhos futuros, dentre citamos: 1) Verificar se a utilização de outros modelos de Arduinos podem influenciar na coleta das assinaturas de identificação; 2) Avaliar se a temperatura do cartão microSD pode influenciar na coleta das assinaturas; 3) Verificar se há um limite de coleta para esse tipo de extração utilizado nesse trabalho; 4) Analisar se o perceptron multicamadas pode generalizar com números maiores de classes, assim representando vários cartões microSD; 5) Analisar se a extração de assinaturas podem ser coletadas em outras plataformas como, por exemplo, a *interface* de cartão SD do próprio computador. E se nessa nova *interface* as assinaturas de identificação são mantidas; 6) Comparar o perceptron multicamadas com outros algoritmos para validação das assinaturas; 7) Por fim, propor uma abordagem de assinaturas de identificação para componentes com *IoT*.

Referências

ALMEIDA, J. de et al. *Dashboardsmartroom: web das coisas para gerenciamento de salas em um campus universitário*. [S.l.]: WebMedia, 2015. Citado na página 14.

BABAR, S. et al. Proposed embedded security framework for internet of things (iot). In: IEEE. *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*. [S.l.], 2011. p. 1–5. Citado 2 vezes nas páginas 15 e 16.

BANZI, M.; SHILOH, M. *Primeiros Passos com o Arduino–2ª Edição: A plataforma de prototipagem eletrônica open source*. [S.l.]: Novatec Editora, 2015. Citado na página 25.

CAULFIELD, A. M. et al. Characterizing flash memory: Anomalies, observations, and applications. Citeseer, 2009. Citado 3 vezes nas páginas 23, 27 e 28.

COLOURIS, G.; DOLLIMORE, J.; KINDBERG, T. Sistemas distribuídos: conceitos e projetos. *Porto Alegre: Bookman*, 2007. Citado na página 15.

DOUKAS, C. *Building Internet of Things with the ARDUINO*. [S.l.]: CreateSpace Independent Publishing Platform, 2012. Citado na página 14.

ELECTRO SCHEMATICS. *Arduino Mega 2560 Pinout*. 2014. Disponível em: <https://www.electroschematics.com/arduino-mega-2560-pinout/>. Acesso em: Accessed: 2019-10-11. Citado na página 30.

FERREIRA, H. G. C. Arquitetura de middleware para internet das coisas. 2014. Citado na página 14.

FLORENCIO, F. d. A. et al. Intrusion detection via multilayer perceptron using a low power device. In: ACM. *Proceedings of the Euro American Conference on Telematics and Information Systems*. [S.l.], 2018. p. 11. Citado na página 26.

GOODRICH, M. T.; TAMASSIA, R. *Introdução à segurança de computadores*. [S.l.]: Bookman, 2013. Citado na página 15.

GUIMARÃES, P. R. B. Análise de correlação e medidas de associação. *Universidade Federal do Paraná. Disponível em:< https://docs. ufpr. br/˜ jomarc/correlacao. pdf>. Acesso em*, v. 9, 2017. Citado na página 36.

HELFMEIER, C. et al. Cloning physically unclonable functions. In: IEEE. *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. [S.l.], 2013. p. 1–6. Citado na página 19.

HRIBERNIK, K. A. et al. Co-creating the internet of things—first experiences in the participatory design of intelligent products with arduino. In: IEEE. *Concurrent Enterprising (ICE), 2011 17th International Conference on*. [S.l.], 2011. p. 1–9. Citado na página 14.

JING, Q. et al. Security of the internet of things: Perspectives and challenges. *Wireless Networks*, Springer, v. 20, n. 8, p. 2481–2501, 2014. Citado na página 16.

KELLER, C. et al. Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers. In: IEEE. *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. [S.l.], 2014. p. 2740–2743. Citado na página 18.

LI, Y. et al. Multilayer perceptron method to estimate real-world fuel consumption rate of light duty vehicles. *IEEE Access*, IEEE, v. 7, p. 63395–63402, 2019. Citado na página 35.

LIU, J.; XIAO, Y.; CHEN, C. P. Authentication and access control in the internet of things. In: IEEE. *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*. [S.l.], 2012. p. 588–592. Citado 2 vezes nas páginas 15 e 19.

MA, Z. et al. Decorrelation of neutral vector variables: Theory and applications. *IEEE transactions on neural networks and learning systems*, IEEE, v. 29, n. 1, p. 129–143, 2016. Citado na página 26.

MARTINS, R. d. S. O. Aplicação da mineração de dados para a recomendação de parâmetros para o coin-or branch and cut. 2016. Citado na página 36.

MOROZOV, S.; MAITI, A.; SCHAUMONT, P. An analysis of delay based puf implementations on fpga. In: SPRINGER. *International Symposium on Applied Reconfigurable Computing*. [S.l.], 2010. p. 382–387. Citado na página 17.

NUNES, A. L. Um estudo investigativo de algoritmos de regressão para data streams. Universidade do Vale do Rio dos Sinos, 2017. Citado na página 36.

PRABHU, P. et al. Extracting device fingerprints from flash memory by exploiting physical variations. In: SPRINGER. *International Conference on Trust and Trustworthy Computing*. [S.l.], 2011. p. 188–201. Citado 8 vezes nas páginas 17, 18, 20, 23, 24, 25, 26 e 28.

RAHMATI, A. et al. Probable cause: the deanonymizing effects of approximate dram. *ACM SIGARCH Computer Architecture News*, ACM, v. 43, n. 3, p. 604–615, 2016. Citado 4 vezes nas páginas 23, 25, 27 e 28.

RESPONSE, S. *Regin: Top-tier espionage tool enables stealthy surveillance*. [S.l.], 2014. Citado na página 16.

SANTOS, B. P. et al. Internet das coisas: da teoriaa prática. 2016. Citado na página 16.

SILVA, C. H. F.; PRAZERES, C. V. Barramento de serviços para publicação de dispositivos na web das coisas. In: *X Workshop de Trabalhos de Iniciação Científica em Sistemas Multimídia e Web*. [S.l.: s.n.], 2013. Citado na página 14.

SKOROBOGATOV, S. P. *Semi-invasive attacks: a new approach to hardware security analysis*. Tese (Doutorado) — University of Cambridge Ph. D. dissertation, 2005. Citado 7 vezes nas páginas 16, 17, 18, 19, 22, 27 e 28.

SKOROBOGATOV, S. P.; ANDERSON, R. J. Optical fault induction attacks. In: SPRINGER. *CHES*. [S.l.], 2002. v. 2523, p. 2–12. Citado 2 vezes nas páginas 27 e 28.

SOLIMAN, M. et al. Smart home: Integrating internet of things with web services and cloud computing. In: IEEE. *2013 IEEE 5th international conference on cloud computing technology and science*. [S.l.], 2013. v. 2, p. 317–320. Citado na página 25.

STALLINGS, W. *Arquitetura e organização de computadores*. Pearson, 2010. ISBN 9788576055648. Disponível em: <https://books.google.com.br/books?id=kTKeQwAACAAJ>. Citado na página 23.

SUH, G. E.; DEVADAS, S. Physical unclonable functions for device authentication and secret key generation. In: ACM. *Proceedings of the 44th annual design automation conference*. [S.l.], 2007. p. 9–14. Citado 6 vezes nas páginas 16, 17, 18, 19, 23 e 26.

SUTAR, S.; RAHA, A.; RAGHUNATHAN, V. Memory-based combination pufs for device authentication in embedded systems. *IEEE Transactions on Multi-Scale Computing Systems*, IEEE, v. 4, n. 4, p. 793–810, 2018. Citado 3 vezes nas páginas 17, 19 e 26.

TANENBAUM, A. S. Redes de computadores quarta edição. *Editora Campus*, 2003. Citado na página 15.

TEHRANIPOOR, F. et al. Dram based intrinsic physical unclonable functions for system level security. In: ACM. *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. [S.l.], 2015. p. 15–20. Citado 6 vezes nas páginas 18, 19, 20, 23, 25 e 28.

TOCCI, R.; WIDMER, N.; MOSS, G. *SISTEMAS DIGITAIS - PRINCIPIOS E APLICAÇOES*. PEARSON BRASIL, 2007. ISBN 9788576059226. Disponível em: <https://books.google.com.br/books?id=KI8DaAEACAAJ>. Citado na página 24.

WANG, Y. et al. Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints. In: IEEE. *Security and Privacy (SP), 2012 IEEE Symposium on*. [S.l.], 2012. p. 33–47. Citado 3 vezes nas páginas 23, 26 e 28.

WANGHAM, M. S.; DOMENECH, M. C.; MELLO, E. R. de. Infraestrutura de autenticaçao e de autorizaçao para internet das coisas. *Minicursos do XIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais—-SBSeg*, v. 2013, p. 156–205, 2013. Citado na página 15.

WITTEN, I. H. et al. *Data Mining: Practical machine learning tools and techniques*. [S.l.]: Morgan Kaufmann, 2016. Citado na página 35.

ZHANG, Z.-K. et al. Iot security: ongoing challenges and research opportunities. In: IEEE. *2014 IEEE 7th international conference on service-oriented computing and applications*. [S.l.], 2014. p. 230–234. Citado na página 16.