UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO CAMPUS CARAÚBAS
CURSO DE ENGENHARIA ELÉTRICA

ÁKIRA COUZACK DE FREITAS COSTA

ANÁLISE COMPARATIVA ENTRE MÉTODOS DE CRIPTOGRAFIA APLICADA A dsPIC

CARAÚBAS – RN
2021

ÁKIRA COUZACK DE FREITAS COSTA

ANÁLISE COMPARATIVA ENTRE MÉTODOS DE CRIPTOGRAFIA APLICADA A dsPIC

Monografia apresentada a Universidade Federal Rural do Semiárido UFERSA, como requisito para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Me. Antônio Alisson Alencar Freitas.

Co-orientador: Prof. Dr. Francisco de Assis Brito Filho

CARAÚBAS -RN

2021

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Central Orlando Teixeira (BCOT)
Setor de Informação e Referência (SIR)

C837a COSTA, AKIRA COUZACK DE FREITAS.
Análise comparativa entre métodos de criptografia aplicada a dsPIC / AKIRA COUZACK DE FREITAS COSTA. - 2021.
58 f. : il.

Orientador: Antônio Alisson Alencar Freitas.
Coorientador: Francisco de Assis Brito Filho.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia Elétrica, 2021.

1. Criptografia. 2. Segurança da informação. 3. dsPIC. I. Alencar Freitas, Antônio Alisson , orient. II. Brito Filho, Francisco de Assis , co-orient. III. Título.

Bibliotecário-Documentalista
Nome do profissional, Bib. Me. (CRB-15/10.000)

ÁKIRA COUZACK DE FREITAS COSTA

ANÁLISE COMPARATIVA ENTRE MÉTODOS DE CRIPTOGRAFIA APLICADA A dsPIC

Monografia apresentada a Universidade Federal Rural do Semiárido UFERSA, como requisito para obtenção do título de Bacharel em Engenharia Elétrica

Defendida em: _____ / _____ / 2 _____.

BANCA EXAMINADORA

_____
Prof. Me. Antônio Alisson Alencar Freitas - UFERSA
Presidente

_____
Prof. Dr. Francisco de Assis Brito Filho - UFERSA
Membro Examinador

_____
Prof. Dr. Hugo Maia - UFERSA
Membro Examinador

_____
Prof. Dra. Mariana de Brito Maia - UFERSA
Membro Examinador

_____
Prof. Dr. Tony Kleverson Nogueira - UFERSA
Membro Examinador

AGRADECIMENTOS

Primeiramente agradeço a Deus por me guiar por todos esses anos sempre me dando forças para continuar e enfrentar todos os obstáculos.

Agradeço à minha avó Maria da Saúde de Oliveira, ao meu pai Antônio Eron da Costa, à minha mãe Antônia Cláudia de Freitas Apolinário e a todos os outros familiares por todo apoio, esforço, carinho e dedicação que tiverem por mim ao longo de todos os anos desde meu nascimento, buscando sempre me motivar a seguir em frente na conquista de meus objetivos.

À minha namorada, Maria Victória, por toda paciência, por estar sempre a meu lado em todos os momentos, por sempre me motivar a fazer o melhor e por toda ajuda durante a realização deste trabalho, tanto pelo apoio quanto nas correções textuais.

Aos meus orientadores Prof. Me. Antônio Alisson Alencar Freitas e o Prof. Dr. Francisco de Assis Brito Filho por todos os incentivos e ensinamentos que deram suporte para meu crescimento acadêmico. Assim como também aos professores, a Dra. Mariana de Brito Maia e o Dr. Tony Kleverson Nogueira.

A todos os meus amigos que contribuíram para meu crescimento como humano e por sempre me apoiarem a conquistar meus objetivos.

À universidade, os professores que foram os pilares necessários para construção do meu conhecimento e aos servidores que com seus esforços fazem a UFERSA.

“Uma criptografia robusta e capaz de resistir a uma aplicação ilimitada de violência. Nenhuma força opressora poderá resolver uma equação matemática”.

Julian Assange

RESUMO

A necessidade de manter informações em segredo sempre foi um desafio a ser superado, tanto pelo estado quanto setores privados, ou até por usuários domésticos. Essa necessidade vem aumentando à medida que surgem novas formas de se conectar e compartilhar informações, tais como em aplicações bancárias, redes sociais ou até em aplicações em Internet das Coisas, e tudo isso aliado ainda aos novos dispositivos que são desenvolvidos para dar suporte às novas aplicações que surgem, como os microcontroladores. Nesse contexto, o foco desse trabalho foi avaliar o desempenho de criptografias aplicadas em um microcontrolador da família dsPIC. Para esse fim foram escolhidas quatro cifras, as cifras Afim e Vigenère, pelo seu valor didático e histórico, e duas outras cifras populares, a cifra assimétrica RSA e a simétrica TEA. Os testes foram realizados a partir da comunicação entre o dsPIC 33EP512MU810 e um ESP32 via comunicação serial assíncrona. A avaliação foi feita a partir de quatro parâmetros, sendo estes o uso de memória ROM, RAM, tempo de processamento e o consumo de energia gasta no processo. Dentre as quatro cifras, as cifras Afim e Vigenère se mostraram bastante eficientes quando comparadas as outras mostrando que seu ponto fraco de fato é a facilidade de serem quebradas. Já as cifras RSA e TEA apresentaram desempenhos parecidos sendo a TEA mais eficiente em termos de processamento, no entanto é menos segura que a RSA.

Palavras-chave: Criptografia. dsPIC, Segurança da informação.

ABSTRACT

The need to keep information secret has always been a challenge to be overcome, both by the state and private sectors or even by home users. This need is increasing as new ways of connecting and sharing information emerge, such as in banking applications, social networks, or even in Internet of Things applications, and all of this allied to the new devices that are developed to support the new applications that arise, such as microcontrollers. In this context, the focus of this work was to evaluate the performance of encryptions applied to a microcontroller in the dsPIC family. For this purpose, four ciphers were chosen, Affine and Vigenère ciphers, for their didactic and historical value, and two other popular ciphers, the asymmetric RSA and the symmetric TEA. The tests were performed based on the communication between the dsPIC 33EP512MU810 and an ESP32 via asynchronous serial communication. The evaluation was made based on four parameters, these being the use of ROM memory, RAM, processing time, and the consumption of energy spent in the process. Among the four ciphers, at Affine and Vigenère cipher proved to be quite efficient when compared to the others, showing that their weak point is in fact the ease of being broken. The RSA and TEA ciphers showed similar performances, with TEA being more efficient in terms of processing, however, it is less secure than RSA.

Keywords: Cryptography. dsPIC, Information security.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	Analog Digital Converter
ANSI	American National Standards Institute
CAN	Controller Area Network
CPU	Central Process Unit
DAC	Digital Analog Converter
DC	Direct current
DES	Data Encryption Standard
DSP	Digital Signal Processor
EM	Enable
I2C	Inter Integrated Circuit
IDE	Integrated Development Environment
IOT	Internet of Things
MDC	Máximo Divisor Comum
PWM	Pulse Width Modulation
RAM	Random Access Memory
RFID	Identificação por radiofrequência
ROM	Read Only Memomy
SPI	Serial Peripheral Interface
TEA	Tiny Encryption Algorithm
TFT	Thin Film Transistor
UART	Universal Asynchrounous Receiver Transmiter
USB	Universal Serial Bus
SCK	Serial Clock
SCL	Serial Clock
SDA	Serial Data
SDI	Slave Data IN
SDO	Slave Data OUT

SUMÁRIO

1 INTRODUÇÃO

A troca de informação constante, onde uma massiva quantidade de dados, que são trocados por setores públicos e privados ou mesmo informações pessoais, são transmitidos através de protocolos de comunicação que podem estar sendo interceptados por terceiros, gera uma necessidade de garantir a segurança e privacidade da informação. Então se fez necessária a criação da segurança da informação, que tem como objetivo proteger os dados a fim de manter sua privacidade e segurança e, para esse fim, é necessário manter a integridade da informação, a confidencialidade acerca do conteúdo e a disponibilidade, sendo acessível a qualquer momento por aqueles que têm permissão para isso (ALBARELLO, 2019).

Uma maneira de manter a segurança dos dados é a partir do uso de protocolos criptográficos que, a partir do uso de algoritmo que realiza diversos cálculos matemáticos ao redor de um valor específico escolhido chamado de chave, embaralha uma informação de modo a tornar ilegível a qualquer um que não seja o remetente ou destinatário da informação, e a única maneira de tornar a informação legível novamente é a partir do uso de uma chave, que contém um valor específico tal que consegue reverter os cálculos.

Em 2019 uma pesquisa feita pela NCIPHER, sobre como a criptografia avançou nos últimos 14 anos, mostrou que desde 2015 a empresa, de modo geral, vem adotando cada vez mais estratégias quanto à criptografia dos seus dados. Dos 5.856 indivíduos entrevistados em 14 países, incluindo o Brasil, 45 % afirmaram que suas empresas adotam constantemente estratégias quanto a segurança da informação, enquanto que 42 % disseram que tinha planos bem limitados quanto a segurança dos dados (PONEMON INSTITUTE, 2019).

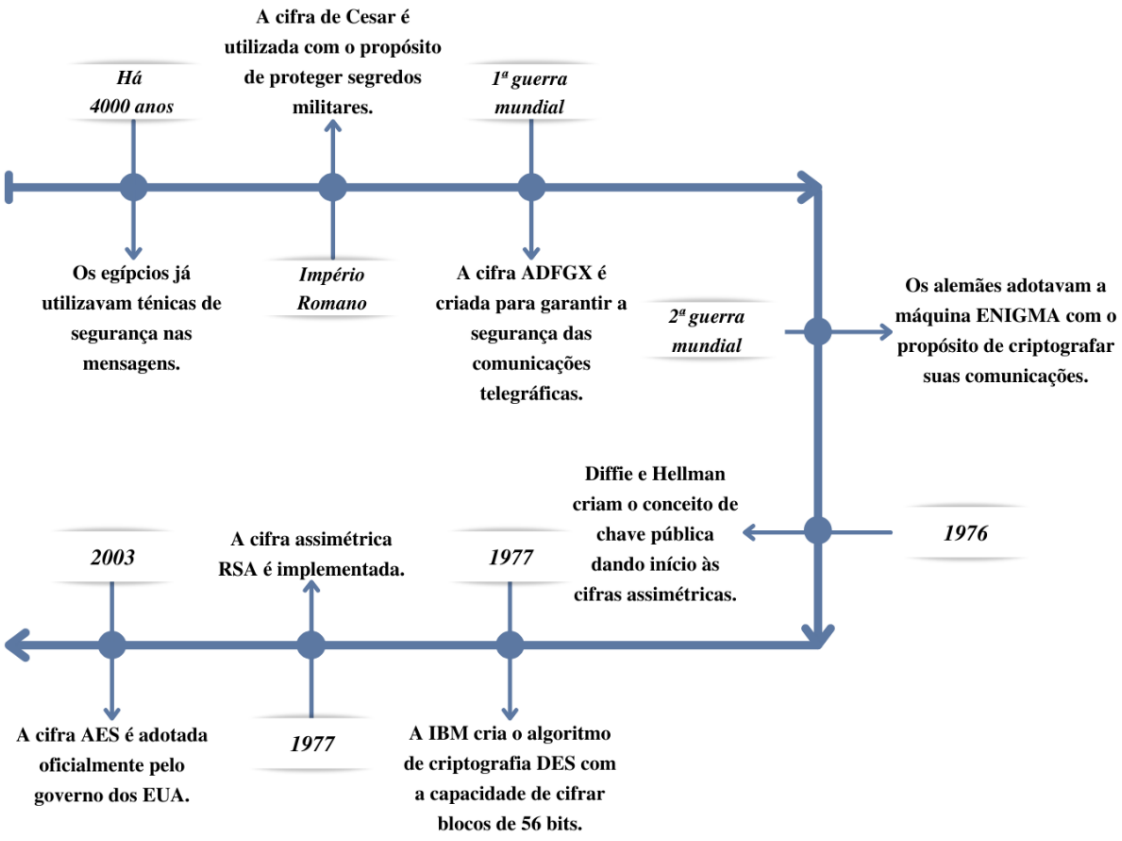
Com os avanços alcançados referentes a serviços via internet, como nuvem ou mesmo a internet das coisas ou apenas IOT, a criptografia como forma de proteção desses serviços se tornou prioridade número 1 para 69 % dos entrevistados, segundo o estudo apresentado pelo Ponemon Institute (2019).

Uma criptografia pode ou não ter uma chave pública, que é compartilhada pelo remetente e destinatário, em posse dessa chave e da chave privada, que pertence apenas a quem faz parte do processo de comunicação, é possível acessar um dado por exemplo e, nesse caso, o algoritmo de criptografia é assimétrico. Caso seja exigida apenas uma chave para cifrar e decifrar uma informação, então esse algoritmo é simétrico.

A Figura 1 mostra, de forma resumida, uma linha do tempo dos principais eventos acerca da criptografia. As primeiras aplicações de técnicas de comunicação segura remetem aos

egípcios que aplicavam os hieróglifos. Outros povos que utilizaram bastante cifras para a proteção de segredos, principalmente militares, foram os romanos. A cifra de Cesar, ou de substituição, leva este nome como homenagem a Júlio Cesar que usava constantemente a cifra de substituição para se comunicar com outros generais (SILVA, 2019).

Figura 1 – Linha do tempo da criptografia.



Fonte: Elaborada pelo autor.

Durante a primeira guerra mundial a junção das técnicas de substituição e sobreposição fomentaram a criação da cifra ADFGX, que era aplicada na proteção das transmissões telegrafadas. Essa técnica era uma modificação do quadrado de Polybius, que utilizava números ao invés das letras A, D, F, G e X (SILVA, 2019). Porém, foi na segunda guerra mundial que houve o grande salto das técnicas de segurança, quando o exército alemão adotou a máquina Enigma como padrão de comunicação. A Enigma (Figura 2) era uma máquina eletromecânica que contava com até oito rotores e, a partir do movimento destes, eram alcançadas as diferentes combinações de encriptação (CASTELLÓ; VAZ, 2021).

Figura 2 – Máquina Enigma.

Fonte: REVISTA GALILEU, 2021.

Em 1976 Diffie e Hellman criaram o conceito de chave pública, que deu o ponta pé inicial para as cifras assimétricas. O sistema Diffie-Hellman não era capaz de cifrar informações, porém seu propósito era que dois indivíduos fossem capazes de entrar em acordo com a chave pública ser utilizada.

Um ano depois a IBM desenvolveu a cifra Data Encryption Standard (DES), de chave simétrica de 56 bits e com capacidade de permitir algo em torno de 72 quadrilhões de combinações. Foi bastante utilizada pelo governo dos Estados Unidos até ser substituída pela cifra DES de 128 bits em 2003 (OLIVEIRA, 2012). Ainda em 1977 o algoritmo RSA foi criado por Ron Rivest, Adi Shamir e Len Adleman. A cifra RSA é até hoje o sistema de chave pública mais utilizado, sendo também o mais seguro entre as cifras assimétricas (OLIVEIRA, 2012).

Atualmente existem diversos protocolos de criptografia com foco na transmissão segura de dados, seja em computadores tais como os domésticos de uso geral ou sistemas embarcados como os microcontroladores tais como a família de dispositivos dsPIC, inclusive atualmente parte dos algoritmos utilizados são projetados com a finalidade de serem mais leves, em termos de processamento e uso de memória, para se adequarem aos dispositivos aplicados em sistemas embarcados (SILVA, 2019).

Um dos desafios enfrentados pelos sistemas embarcados como os microcontroladores está ligado à garantia que as informações transmitidas sejam seguras e privadas, sendo essa uma das principais preocupações demonstradas por governos e empresas pelo mundo. Essa necessidade de sistemas que protejam a privacidade e segurança dos dados que estão sendo compartilhados a partir de sistemas que utilizam microcontroladores vem crescendo e, em conjunto, cresce também a demanda por técnicas de criptografia que sejam aplicadas de forma

que se adeque as limitações de hardware tais como a quantidade de memória, velocidade do processador e a energia gasta no processo.

Portanto, este trabalho tem como proposta implementar algoritmos criptográficos como medida de segurança da informação, com a finalidade de analisar e comparar os desempenhos obtidos por cada um, para desse modo apontar quais são as que melhor se adequam ao microcontrolador.

Para uma melhor compreensão do trabalho, a organização do mesmo é feita de maneira que no capítulo 2 é abordado um referencial teórico, feito com base em pesquisas bibliográficas acerca da segurança da informação, tendo ênfase na criptografia. Em seguida, no capítulo 3 Já são abordados os dispositivos utilizados para os testes propostos. No capítulo 4 é abordada a metodologia empregada no trabalho. O capítulo 5 mostra os resultados obtidos e discussões a respeito deles. Por fim, os capítulos 6 e 7 são, respectivamente, as conclusões obtidas e as referências utilizadas.

2 SEGURANÇA DA INFORMAÇÃO

A crescente demanda por dispositivos que se conectem à rede mundial de computadores e pela troca de informação, no intuito de executar aplicações do dia a dia, tende a resultar em problemas de privacidade e segurança das informações trocadas. A preocupação quanto a segurança dos dados transmitidos vem sendo considerada um desafio a ser vencido por diversas empresas, no contexto da espionagem industrial, ou por parte dos governos, com o objetivo de manter seus segredos para si (LEMOS JUNIOR, 2018).

A segurança da informação pode ser vista como uma maneira de realizar a proteção de um grupo de informações e tem como objetivo defendê-las garantindo os quatro pilares que são (LEMOS JUNIOR, 2018):

- Autenticidade, que deve garantir que o remetente/destinatário da informação possa verificar a identidade do remetente/destinatário, garantindo assim a veracidade da informação;
- Confidencialidade, de forma que a informação só possa ser acessada por quem tem autorização;
- Disponibilidade, que deve garantir a acesso ao documento afim de que não ocorram problemas ligados à queda de sistemas, seja ou não intencional;
- Integridade, que está ligada a capacidade de a informação transmitida poder ser checada pelo destinatário, evitando falsificação ou adulteração da informação;

Dentre as várias ferramentas utilizadas com o foco na segurança da informação, atualmente as mais aplicadas estão ligadas à criptografia. O termo criptografia vem da junção de duas palavras gregas *kryptos*, que significa ocultar, e *grafos*, que significa escrever, juntas a "escrita oculta" faz parte da área de estudos da criptologia, que tem como objeto de estudo a comunicação segura (ESPINDOLA, 2017).

A criptografia funciona embaralhando uma informação que se deseja proteger. O processo ocorre utilizando funções matemáticas complexas, afim de garantir que estas não sejam reversíveis se não houver uma chave, que é a única maneira viável de restaurar a mensagem original.

Um outro termo interessante a ser abordado é a criptoanálise, que tem como definição a tentativa de quebrar textos cifrados afim de descobrir como o algoritmo por traz da criptografia funciona (ALMEIDA, 2012). Além da criptoanálise outras duas técnicas são utilizadas, a tentativa de quebrar criptografias por tentativa e erro é chamada de força-bruta e, dependendo

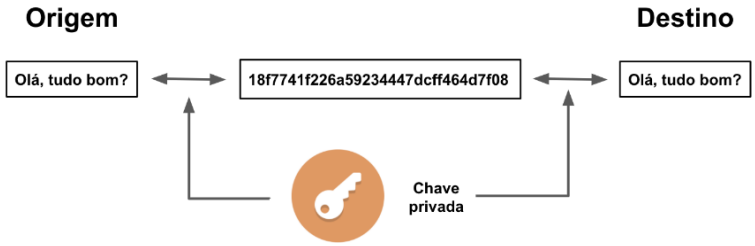
da complexidade, é possível levar muito tempo para decifrar a mensagem, tornando o processo muito caro computacionalmente. A segunda técnica é a análise de frequência em que, a partir da análise do texto cifrado, é possível analisar as letras mais populares dentro da linguagem em questão, e então comparando com letras que mais aparecem no texto cifrado, é possível descobrir o texto original (NAVARRO, 2014).

Os algoritmos de criptografia podem ainda ser classificados quanto à forma de processamento da informação em algoritmos de fluxo e de bloco. As cifras de fluxo realizam o processo de encriptação bit a bit ou byte a byte geralmente são mais rápidas e apresentam um baixo índice de erro já que o processo e realizado símbolo a símbolo, enquanto as cifras de bloco trabalham com pacotes de bytes, geralmente em torno de 64 a 128 bits. Sendo assim, as cifras de blocos têm uma alta difusão, ou seja, a encriptação depende de vários símbolos e não de apenas um, porem os algoritmos de blocos são mais lentos e tem um maior índice de erro já que todo o bloco está ligado ao processamento de um símbolo (SERAFIM, 2012).

2.1 Criptografia Simétrica

As primeiras criptografias utilizadas eram simétricas e utilizavam uma senha, que era compartilhada por todos que estão no extremo da comunicação. A senha ou a chave privada é única, servindo para cifrar ou decifrar (SILVA, 2019). Na figura 3 pode ser visto um exemplo de comunicação segura utilizando uma criptografia de chave privada.

Figura 3 – Fluxo de uma cifra Simétrica.



Fonte: SAKURAI, 2020.

As cifras simétricas são mais simples de serem implementadas por exigirem menos cálculos em relação às cifras de chave pública. As principais desvantagens do uso de criptografias simétricas é que a mesma chave pode cifrar e decifrar uma mensagem e, por isso,

tem um grande potencial de ser descoberta, tornando o canal vulnerável. Outro problema está na quantidade de chaves necessárias que aumentam a cada participante. A quantidade de chaves cresce *n(n - 1)/2*, onde n é o número de participantes (SILVA, 2019).

A cifra de César é um exemplo de criptografia simétrica e foi bastante utilizada no passado, para proteger informações sensíveis em tempos de guerra. Atualmente a cifra de César é considerada obsoleta, porém ela tem um grande valor didático, dada sua simplicidade, por isso ela será explicada aqui, apenas por sua importância histórica e didática (PEREIRA, 2015).

A cifra é baseada no processo de substituição de cada caractere da mensagem por um outro valor. O processo de substituição consiste em uma chave que tem um valor numérico e a partir da chave, é realizado o deslocamento da mensagem, substituindo cada letra pelo seu equivalente para a chave escolhida (PEREIRA, 2015). Por exemplo, tomando a mensagem "ATAQUEM PELO NORTE" e escolhendo a chave com o valor 3, então a mensagem cifrada é vista na figura 4:

Figura 4 – Mensagem cifrada.

A	T	A	Q	U	E	M	P	E	L	O	N	O	R	T	E
D	W	D	T	X	H	P	S	H	O	R	Q	R	U	W	H

Fonte: Elaborada pelo autor.

O processo para decifrar a informação se dá deslocando a mensagem codificada no sentido oposto o mesmo número de vezes da chave definida (SILVA, 2019).

Uma melhor forma de visualizar a cifra de Cesar é a partir de equações matemáticas. A equação 1 mostra como encriptar os dados, *C(i)* é o símbolo encriptado, a *m* é a mensagem e a é *a* chave. Como o alfabeto latino utiliza 26 símbolos, então é necessário o uso do "mod 26", uma artificio da matemática modular que de forma simplificada caso o número seja maior que 26, será então considerado o resto da divisão entre o número e 26, e assim garante que o resultado seja um dos símbolos do alfabeto. A equação 2 é o processo para decifrar, onde é feito o inverso, nesse caso a subtração da mensagem cifrada da chave usada para cifrar.

$$C(i) = m + a \ (mod 26) \tag{1}$$

$$D(i) = C(I) - a \ (mod26) \quad (2)$$

A cifra afim, assim como a de Cesar, utiliza o processo de substituição. O nome afim é por ser semelhante à função afim, onde, a chave agora é formada por dois números *a* e *b* inteiros, de modo que *a* e *b* são positivos e que o máximo divisor comum (MDC) entre *a* e 26 é 1. A expressão que executa o processo de codificação é dada por:

$$D(i) = aC(i) + b \ (mod26) \quad (3)$$

onde *m* é a mensagem original a ser embaralhada e *C(i)* é a mensagem embaralhada (GALDINO, 2014).

Utilizando a palavra "MENSAGEM" como exemplo e os valores de *a* = 2 e *b* = 5. Tomando cada letra do alfabeto como um número onde A equivale a 1, B = 2, ..., Z = 26, então é obtida a mensagem convertida, conforme mostra a figura 5:

Figura 5 – Mensagem transformada em números.

M	E	N	S	A	G	E	M
13	5	14	19	1	7	5	13

Fonte: Elaborada pelo autor.

Agora, utilizando a expressão 1.0 para cifrar a mensagem:

- $C(13) = 2 \cdot 13 + 5 = 5(mod26);$
- $C(5) = 2 \cdot 5 + 5 = 15(mod26);$
- $C(5) = 2 \cdot 14 + 5 = 7(mod26);$
- $C(19) = 2 \cdot 19 + 5 = 17(mod26);$
- $C(1) = 2 \cdot 1 + 5 = 7(mod26);$
- $C(7) = 2 \cdot 7 + 5 = 19(mod26);$
- $C(5) = 2 \cdot 5 + 5 = 15(mod26);$
- $C(13) = 2 \cdot 13 + 5 = 5(mod26);$

Finalmente a mensagem codificada:

Figura 6 – Mensagem cifrada.

5	15	7	17	7	19	15	5
E	O	G	Q	G	S	O	E

Fonte: Elaborada pelo autor.

O processo para desembaralhar a mensagem codificada se dá na inversa da função afim, logo:

$$C(i) = a^{-1}(i - b)(mod26) \quad (4)$$

As duas cifras explicadas anteriormente eram monoalfabéticas, ou seja, utilizavam um alfabeto de modo que, cada letra na mensagem original era substituída por uma outra durante o processo. Esse tipo de cifra é bastante vulnerável, podendo ser decifrada a partir da análise de frequência ou força bruta (KASPERSKY, 2015). Já a cifra de Vigenère é polialfabética e, assim, utiliza mais de um alfabeto para realizar o processo de encriptação.

A cifra de Vigenère funciona embaralhando a mensagem *i* e a chave *k* que nesse caso não é mais um número e sim um texto. Tomando como exemplo a mensagem *i* = "MENSAGEM" e *k* = "KEY". Como a chave é menor que a mensagem então deve-se repetir a chave *k* até igualar a quantidade de letras da mensagem:

Figura 7 – Mensagem alinhada com a chave.

M	E	N	S	A	G	E	M
K	E	Y	K	E	Y	K	E

Fonte: Elaborada pelo autor.

Como se trata do alfabeto de 26 letras então a quantidade de alfabetos para a substituição deve ser 26 e logo é obtida a tabela da Figura 8 abaixo com todos os alfabetos escolhidos.

Figura 8 – Tabela dos alfabetos definidos.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Fonte: KASPERSKY, 2015.

A partir da tabela da Figura 8 pode ser realizado o processo de substituição, onde as letras da mensagem são representadas pelas colunas e a letras da chave pelas linhas, logo, para o primeiro caso i = M e k = K, então a letra correspondente é W, para i = E e k = E corresponde a I e assim por diante resultando em:

Figura 9 – Mensagem cifrada por Vigenère.

W	I	L	C	E	E	O	Q

Fonte: Elaborada pelo autor.

Para decriptar a mensagem segundo Hunn, Naziri e Idris (2012), basta utilizar a palavra chave e o texto cifrado, buscando na tabela os valores correspondentes, ou seja, olhando para as linhas.

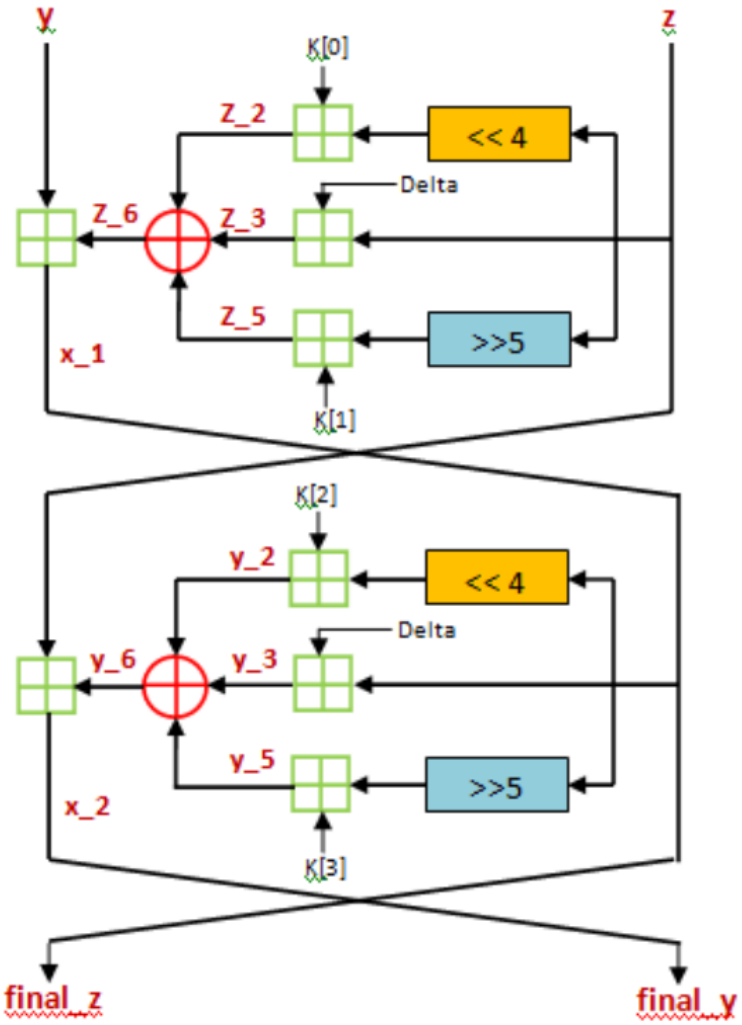
A cifra Tiny Encryption Algorithm (TEA) foi criada por David Wheeler e Roger Needham com o propósito de ser utilizada em dispositivos de diversos tamanhos, já que seu diferencial é o tamanho do espaço que ocupa na memória. A TEA, diferente das cifras já apresentadas, é uma cifra de blocos e, no artigo original escrito pelos criadores da cifra, foi apresentado um código fonte da cifra TEA para uma mensagem separada em dois blocos de 32 bits cada um. Geralmente a cifra é usada com uma chave de 128 bits divido em 4 partes de 32 bits e os processos de encriptação e decriptação são baseados na cifra de Feistel, que utiliza operações de deslocamento duplo de bits com o propósito de misturar os bits da mensagem com os da chave embaralhando seu conteúdo (SHOEB; GUPTA, 2012).

Para melhor entender o processo que está exemplificado na Figura 10 abaixo, tomando uma mensagem *M* de 64 bits dividida em dois blocos de 32 bits sendo *Y* e *Z*, e uma chave *K* de 128 bits dividida em quatro grupos de 32 bits, de modo que se tem *K[0]*, *K[1]*, *K[2]* e *K[3]*. O bloco *Z* é inicialmente deslocado 4 bits para esquerda e então o resultado é somado com *K[0]* e então é encontrado *Z_2*. Depois o bloco *Z* de novo é deslocado, porém agora 5 bits para direita, sendo que o resultado é somado a *K[1]*, e então este pode ser visto como *Z_5*. Finalmente é feita a soma de Z com Delta, que é uma constante derivada da proporção áurea que é 2654435769 ou 0x9E3779B9 em Hexadecimal, e o resultado *Z_3* é adicionado junto com *Z_2* e *Z_5*, através de uma operação XOR, formando *Z_6* que, por fim, é somado ao bloco *Y* gerando *X_1*, encerrando assim metade do ciclo.

A segunda etapa inicia com *X_1* sendo deslocado 4 bits para a esquerda e adicionado a *K[0]*, gerando *Y_2* e, em seguida, *X_1* é deslocado 5 bits para direita e somado a *K[3]*, formando *Y_5*. Então *X_1* agora é somado com Delta para gerar *Y_3* e finalmente, utilizando novamente a porta XOR com *Y_2*, *Y_3* e *Y_5*, é obtido *Y_6* que é somado a *Z*, gerando *X_2*.

O processo é repetido 32 vezes para criptografar por completo a mensagem, que continua dividida em dois blocos representados por *final_Z* e *final_Y*.

Figura 10 – Fluxo da cifra TEA para cifrar.



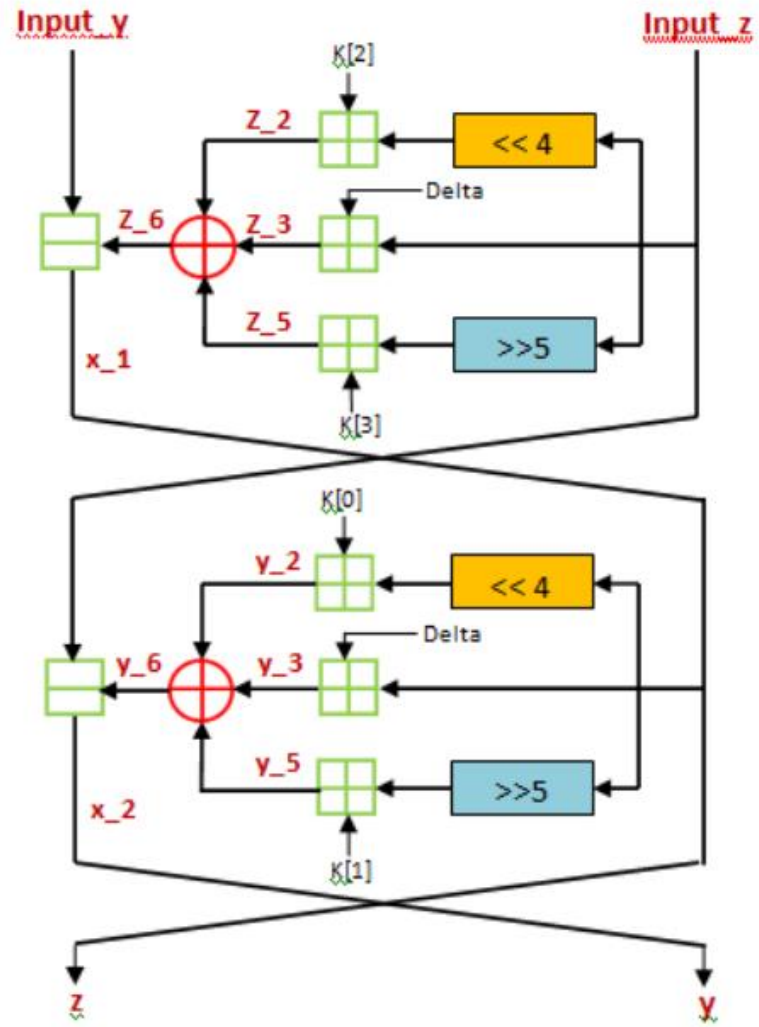
Fonte: HUNN; NAZIRI; IDRIS, 2012.

Para decifrar a mensagem o processo é realizado de maneira semelhante, porém agora as entradas são os valores cifrados, ou seja, *final_Z* que agora é *Input_Z*, e *final_Y*, que é *Input_Y*, conforme visto no fluxograma na figura 11. As chaves também são invertidas e agora o *Input_Z* é deslocado 4 bits para a esquerda e somado com *K[2]*, sendo salvo em *Z_2*, em seguida *Input_Z* é deslocado 5 bits para a direita, somado com *K[3] e* salvo em *Z_5*, depois *Input_Z* é somado a Delta e salvo em *Z_3*. Novamente a porta XOR gera *Z_6*, que é somado a *Input_Y*, formando *X_1*.

A última parte do processo de encriptação é feito com *X_1* deslocado 4 bits para esquerda, somado a *K[0]* e guardado em *Y_2*, em seguida *X_1* é deslocado 5 bits à direita, somado com *K[1]* e é gerado *Y_5*. *X_1* é somado agora a Delta para formar *Y_3* e, por fim, após

usar XOR com *Y_2*, *Y_3* e *Y_5*, forma *Y_6* que, quando somado a *Input_Z*, gera *X_2* retornando ao seu valor inicial *Z* e *Y*.

Figura 11 – Fluxo da cifra TEA para decifrar.



Fonte: HUNN; NAZIRI; IDRIS, 2012.

2.2 Criptografia Assimétrica

As criptografias assimétricas ou de chave pública, utilizam o conceito de chaves pública e privada, onde a chave pública pode ser transmitida de forma livre como o nome explicita, sem agredir os quatro objetivos já citados. Nesse caso, a chave privada é de conhecimento único do proprietário. A Figura 12 mostra um exemplo de comunicação utilizando uma cifra de chave pública. No exemplo, é possível ver a mensagem sendo associada com a chave privada única

do remetente (origem), e então a mensagem é transmitida cifrada até o destinatário que, com uso da chave pública e de sua chave privada, retorna à mensagem original (ANJOS, 2016).

Figura 12 – Fluxo de uma cifra Assimétrica.

Fonte: SAKURAI, 2020.

As cifras assimétricas demandam de uma quantidade maior de processamento computacional, porém, devido a utilização de funções matemáticas complexas, tornam a tentativa de decifrarem por força bruta praticamente impossível. Devido à necessidade de maiores cálculos computacionais, estes algoritmos se tornam caros em termos de processamento em relação às cifras simétricas (ANJOS, 2016).

Uma das principais cifras assimétricas é a RSA, criada por Rivest Shamir e Adleman em 1978 e que ainda nos dias atuais é bastante utilizada. A cifra utiliza a teorias dos números para realizar os processos de encriptação e decriptação e por se trata de uma cifra assimétrica, é necessário que o algoritmo gere duas chaves (CASTRO, 2014).

Uma versão simplificada da cifra RSA segundo visto em Bonfim (2017) segue os seguintes passo:

1. São escolhidos dois números primos aleatórios p e q;
2. É calculado $n = pq$;
3. É calculada a função de $\varphi(n) = (p - 1)(q - 1)$;
4. Em seguida é realizada a escolha de um número inteiro tal que $1 < e < \varphi(n)$, onde e e $\varphi(n)$ são co-primos;
5. É calculado d, de modo que $ed = 1 mod\ \varphi(n)$, e assim, d é o inverso multiplicativo de e;
6. A chave pública é formada pelo par (n, e) e a chave privada pela tripla (p, q, d);

Com as chaves formadas, para realizar a cifragem é só calcular a potência modular utilizando chave pública (n, e) na mensagem m, logo:

$$m^e(i) = c(mod\ n) \quad (5)$$

Já o processo de decodificação da mensagem *m* é feito com a chave privada *(p, q, d)* de tal modo que:

$$c^d(i) = m(mod\ n) \quad (6)$$

2.3 Criptografia Leve

Os dispositivos atuais vêm sendo fabricados em tamanhos reduzidos e isso acaba afetando o poder de processamento, de modo que seja necessário criar métodos mais simples de proteção para esses dispositivos. Para esse fim há o conceito de *lightweight* ou, em português, criptografia leve. Para uma cifra ser considerada leve, ela deve preencher alguns requisitos como:

- Consumo de memória ROM e RAM;
- Baixo consumo de potência;
- Mínimo consumo de energia;
- Velocidade de processamento.

Estes requisitos garantem o correto funcionamento das cifras em dispositivos tais como microcontroladores e até dispositivos de identificação em rádio frequência (RFID), que têm uma quantidade bem limitada de memória quando comparados a computadores, além também de contar com um hardware menos potente, o que impacta na velocidade de processamento da cifra e no consumo de energia. (OKAMURA, 2017)

Nos dias de hoje a quantidade de aplicações que buscam na criptografia a segurança e garantia da privacidade vem aumentando e, graças aos avanços computacionais e aos algoritmos robustos e quase inquebráveis, trazem consigo a possibilidade de pôr em prática diversas aplicações, tais como: sistemas RFID, pagamentos online e, mais recente, as aplicações IOT.

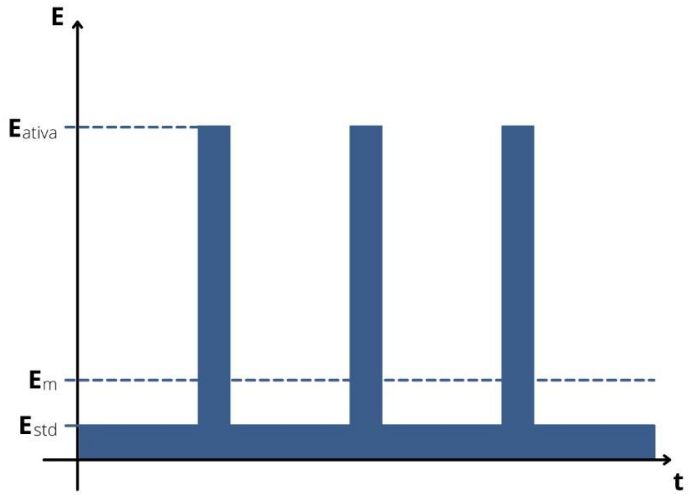
3 DISPOSITIVOS E SENSORES

Por traz das novas aplicações industriais e outras tais como IOT, é necessário que haja um elemento inteligente, capaz de processar as informações adquiridas, com o intuito de devolver os resultados e realizar os processos demandados. Tomando como base a ideia que os dispositivos devem ser compactos a ponto de se tornarem praticamente naturais ao ambiente em que estão inseridos, os candidatos perfeitos para serem o cérebro da aplicação são os microcontroladores, dispositivos tais que apresentam um bom poder de processamento em relação ao seu preço.

Basicamente um microcontrolador é pequeno computador composto de uma Unidade de processamento central ou CPU), memória de dados e programas e periféricos, tais como PWM (Pulse Width Modulation), conversores analógico/digital, controladores de comunicação I2C (Circuito inter-integrado), SPI (Serial Peripheral Interface), USB (Universal Serial Bus) e Ethernet, e ainda pinos de entrada e saída. Sendo que todos esses componentes são fabricados em um chip integrado (MIYADAIRA, 2012).

Os microcontroladores, mais especificamente os processadores não funcionam de forma continua, eles sempre oscilando entre seu funcionamento ativo quando processando alguma informação ou em espera (standby) na forma de uma “hibernação” afim de reduzir o consumo de energia ao desligar partes não essenciais do chip. Na figura 13 o gráfico mostra o exemplo do consumo do processador quando no estado de operação ativo, aumentando o consumo, ou em hibernação, consumindo o mínimo possível e, dessa forma, reduzindo a energia média gasta durante todo o tempo que permanece ligado.

Figura 13 – Gráfico da Energia x tempo de operação do processador.



Fonte: Elaborada pelo autor.

Atualmente existem diversas famílias de microcontroladores, onde estas apresentam arquitetura de hardwares com leve diferença, de modo que apresentam certas características que diferem uma das outras, porém apesar das diferenças, apresentam quase sempre recursos semelhantes (ZELENOVSKY, MEDONÇA, 2017). Dentre as diversas famílias de microcontroladores encontradas no mercado uma das mais utilizadas atualmente é a família dsPIC33 da Microchip, que une o alto desempenho do processamento de sinais, encontrados nos processadores de sinal digital (DSP), e a arquitetura integrada de fácil controle via manipulação de bits dos microcontroladores PIC (MICROCHIP, 2021).

A família dsPIC 33 é projetada em uma arquitetura de 16 bits e apresenta um baixo consumo de energia, além de uma variada quantidade de periféricos de entradas e saídas digitais, analógicas, além de protocolos de comunicação tais como SPI, a universal Asynchrounous Receiver Transmiter (UART), I2C, Controller Area Network (CAN) e Pulse Width Modulation (PWM). Contém também recursos de segurança para garantir a execução de tarefas robustas, conversores analógicos para digital (ADC) e digital para analógico (DAC) de alto desempenho (MICROCHIP, 2021). Na tabela 1 é possível observar alguns dos principais parâmetros dos microcontroladores 33EP512MU810 pertencente à família dsPIC.

Tabela 1 – Parâmetros do dsPIC 33EP512MU810.

Nome	Valor
Arquitetura	16 – bit
Velocidade máxima (CPU)	70 MHz
Velocidade máxima (CPU)	70 MIPS
Memória de programa	512 KB
SRAM	52 KB
Tensão de operação	3.3 V
Corrente de operação	70 mA
Número de pinos	100
UART	4
SPI	4
I2C	2
Temporizadores	9x16 bit 4x32 bit

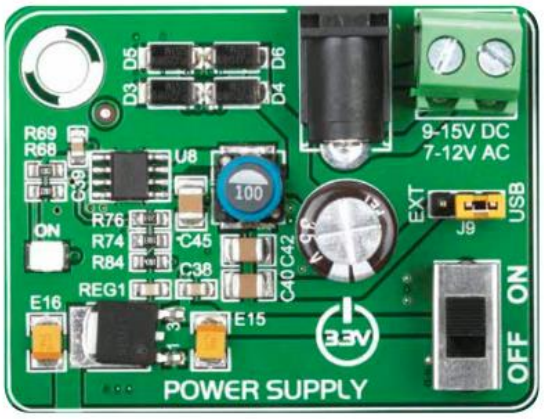
Fonte: MICROCHIP, 2021.

Uma vez que os microcontroladores são apenas chips, se torna problemático trabalhar diretamente com os mesmos no ponto de vista de conectar seus pequenos terminais aos

dispositivos a serem utilizados. Dentre algumas soluções práticas uma que tem grande destaque é a placa de desenvolvimento, que oferece a uma ou mais famílias de microcontroladores uma interface de conexão com os pinos, facilitando a utilização dos mesmos em projetos. Além disso, as placas geralmente oferecem uma gama variada de ferramentas para as mais diversas aplicações, incluindo por exemplo um gravador, que tem a função de escrever o firmware no microcontrolador. No caso da família dsPIC 33, é possível encontrar várias placas sendo uma delas a EasyPIC Fusion V7.

A placa EasyPIC Fusion V7 dá suporte a três famílias de microcontroladores sendo elas a PIC32, PIC24 e dsPIC 33, e conta com um conjunto de módulos que dá suporte ao projeto a aplicações. Dentre os módulos oferecidos pela placa, as que foram utilizadas neste trabalho foram: O Módulo de alimentação, Figura 14, que é equipado com um regulador de tensão e entrega ao sistema 3.3 V CC. O módulo conta ainda com suporte a diversas formas de alimentação podendo ser via USB, sendo alimentado por 5 V CC, ou através de um conector CC, aceitando uma faixa de tensão 7 a 12 V CA ou 9 a 15 V CC, assim como também para o conector parafuso (MATIC, 2013).

Figura 14 – Módulo de alimentação da placa EasyPIC Fusion V7.



Fonte: MIKROELEKTRONIKA (2013).

Para enviar o firmware, a placa conta com o programador MicroProg (Figura 15), equipado com um debugador MikroICD com conexão USB (Universal Serial Bus) 2.0. O programador suporta todos os chips das famílias PIC10, PIC12, PIC16, PIC18, PIC24, dsPIC 30/33, PIC32 (MATIC, 2013).

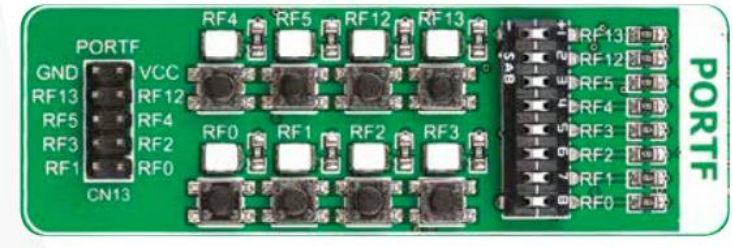
Para a conexão direta com o microcontrolador, a placa conta com um grupo de periféricos de entrada e saída (I/O). Ao todo são nove grupos (portas) de periféricos com até

oito pinos, botões e light-emitting diodes (LED), que estão ligados diretamente às portas I/O do microcontrolador. Conforme pode ser visto na Figura 16, cada grupo conta com dois switches de três estados, que permite escolher tanto se os periféricos da porta serão ativados em nível alto, baixo ou indiferente da tensão, ou se serão pull-up, pull-down ou nenhum. Por fim, também existe um switch de duas posições que habilita ou não os LEDs (MATIC, 2013).

Figura 15 – Módulo de alimentação da placa EasyPIC Fusion V7.

Fonte: MIKROELEKTRONIKA (2013).

Figura 16 – Módulo de alimentação da placa EasyPIC Fusion V7.



Fonte: MIKROELEKTRONIKA (2013).

Quanto à comunicação, a placa conta com quatro modelos USB, sendo dois deles para o protocolo de comunicação serial UART, nesse caso existem a UART A e UART B, que utilizam o padrão B. Um conector USB HOST padrão A, para receber dados de dispositivos como teclados, mouse e outros periféricos. E RF B para conexão com dispositivos host, podendo ser um PC, notebook entre outros (MATIC, 2013).

Um poderoso módulo disponível na placa é o display do tipo Thin Film Transistor (TFT), visto na figura 12, de 320x240 pixels com retro iluminação led capaz de mostrar 262144

cores diferentes. O display TFT conta também com um painel touch de duas camadas resistivas, fazendo da tela TFT um dispositivo de entrada e saída de dados (MATIC, 2013).

Figura 17 – Módulo de alimentação da placa EasyPIC Fusion V7.

Fonte: MIKROELEKTRONIKA (2013).

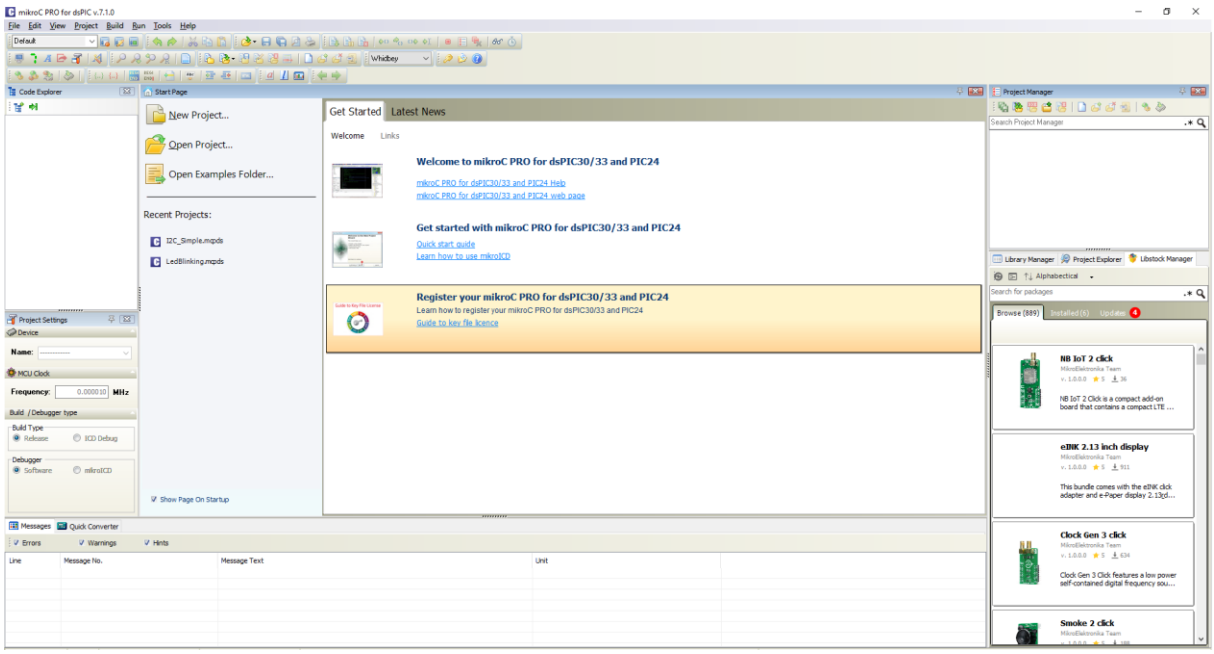
Figura 18 – Módulo TFT.

Fonte: MIKROELEKTRONIKA (2013).

Para programar um dsPIC, assim como qualquer outro microcontrolador, é necessário utilizar um compilador, para que o código escrito em linguagem de programação seja convertido para linguagem de máquina e, em seguida, seja enviado para o programador. Para esse fim há o MikroC PRO que é um compilador ANSI C e que apresenta um conjunto de ferramentas, tais como um ambiente de desenvolvimento integrado (IDE), Figura 19, bibliotecas, otimizadores de hardware e software e ainda, segundo o site da MikroElektronika, suporta mais de 500 chips (BORBA; SILVA, 2016).

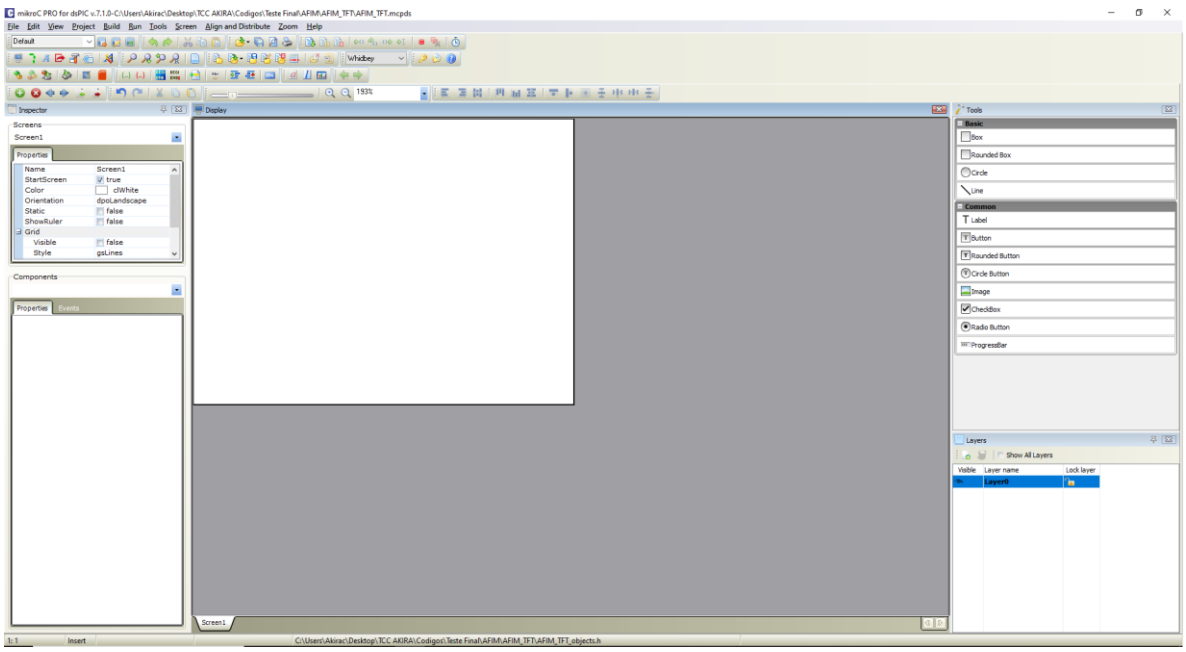
O MikroC ainda oferece algumas ferramentas para dar suporte ao programador como a função Statistics, que gera relatórios acerca da memória utilizada do micro, e também o Visual TFT, Figura 20, que possibilita criar interfaces gráficas para alguns modelos de displays, como o encontrado na placa EasyPIC Fusion V7.

Figura 19 – MikroC PRO for dsPIC.



Fonte: Elaborada pelo autor.

Figura 20 – Visual TFT do MikroC PRO for dsPIC.

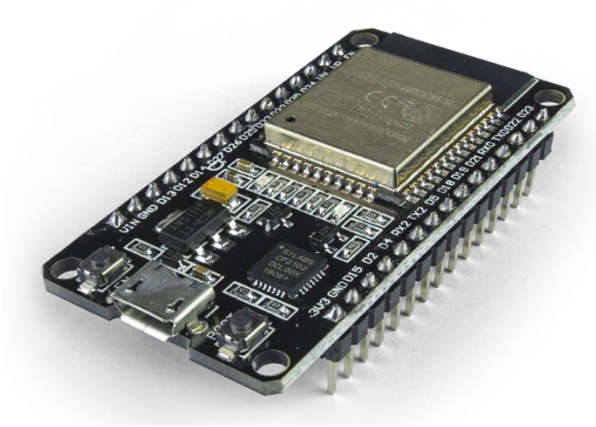


Fonte: Elaborada pelo autor.

A placa de desenvolvimento ESP32, vista na Figura 21, é um dos dispositivos mais utilizados em redes IOT. A placa ESP32-WROOM, modelo que será utilizado neste trabalho, segundo a fabricante ESPRESSIF, conta com um microcontrolador ESP32-D0WD e com

módulos de Wi-Fi e Bluetooth integrados, além de alguns sensores e portas de acesso de entrada e saída.

Figura 21 – ESP32.



Fonte: ROBOCORE, 2021.

A Tabela 2 conta com algumas informações importante retiradas da folha de dados da fabricante.

Tabela 2 – Parâmetros do ESP32.

Nome	Valor
Arquitetura	32 – bit
Velocidade máxima (CPU)	40 MHz
Velocidade máxima (CPU)	40 MIPS
Memória de programa	448 KB
SRAM	520 KB
Tensão de operação	3.3 V
Corrente de operação	80 mA
Número de pinos	30
UART	3
SPI	4
I2C	2
Temporizadores	2 x 64 bits

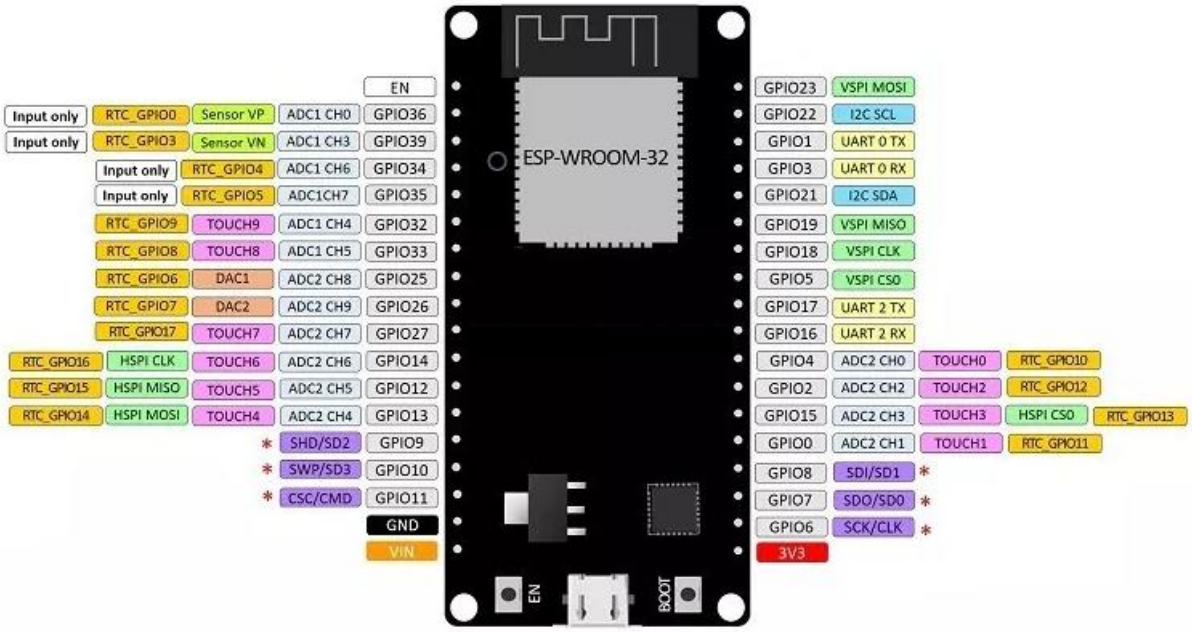
Fonte: adaptado de EPRESSIF (2021).

A alimentação da placa é feita a partir da entrada micro USB, que serve também para a conexão com o computador, afim de ter o firmware implantado no chip. A placa, segundo a

folha de dados, conta com um conjunto de recursos de baixo consumo projetados para otimizar o gasto energético para o mínimo possível durante o funcionamento da placa.

A quantidade de pinos presentes na placa é vista na Tabela 3. Cada porta tem pelo menos uma função sendo que, segundo a Figura 22, alguns pinos chegam até quatro funções. Alguns pinos relevantes para esse trabalho e comunicação de modo geral são os pinos UARTx TX e UARTx RX, para uso na comunicação UART o I2Cx SDA e I2Cx SCL, para comunicação I2Cx, e os pinos SDIx, SDOx e SCKx que servem para comunicação SPI. Os dois botões presentes na placa servem para dar BOOT no sistema e permitir acesso para que o gravador receba o firmware, e o EN que habilita o módulo estando sempre em nível alto.

Figura 22 – Pinagem do ESP32.

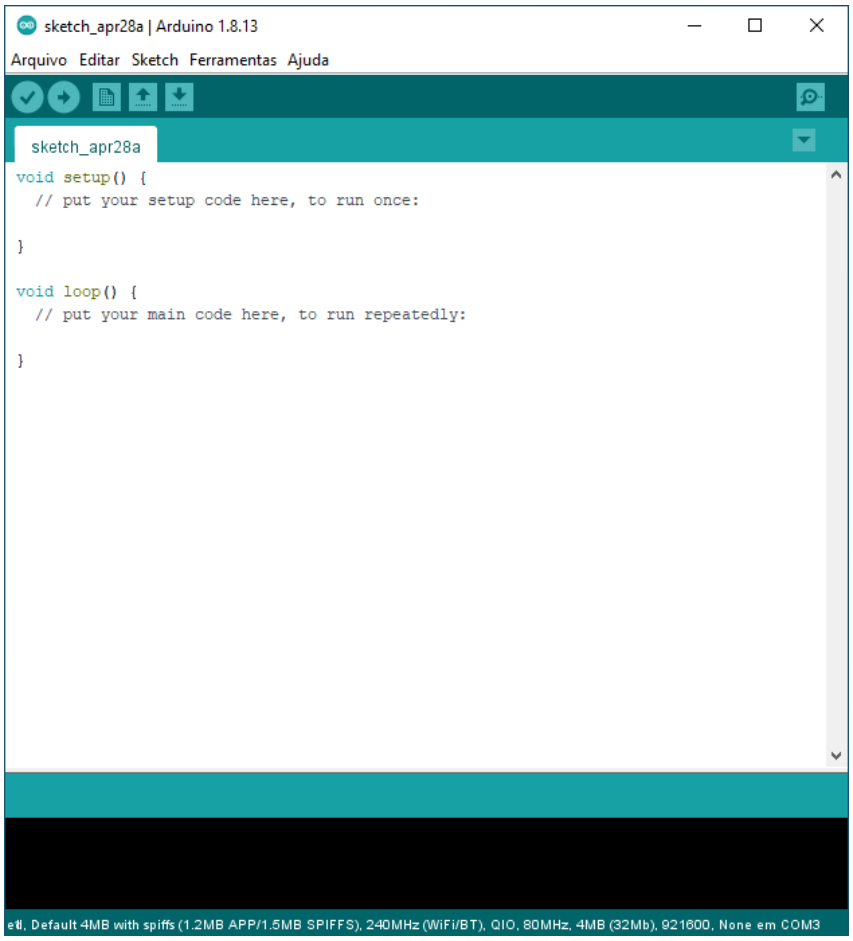


Fonte: CAP SISTEMA, 2020.

Assim como ocorre com o dsPIC, para programar o ESP32 também é necessário um compilador e, consequentemente, um ambiente para o desenvolvimento do mesmo. Segundo o site da plataforma ARDUINO, a IDE Arduino é uma plataforma de software livre para programação em linguagem C/C++, que conta com diversas bibliotecas para expandir as suas funcionalidades. Os programas escritos a partir do Arduino IDE recebem o nome de Sketch e possuem a extensão .ino.

Um sketch é visto na Figura 23 abaixo e, como pode ser visto, tem como padrão duas funções especiais, a função Setup() é chamada uma vez, quando o projeto é executado, e é onde é feita configuração do projeto. A função Loop()é chamada repetidamente durante a execução.

Figura 23 – Aduino IDE.



Fonte: Elaborada pelo autor.

4 METODOLOGIA

O ambiente de testes, criado para a realização das avaliações de desempenhos das cifras, foi feito utilizando o dsPIC 33EP512MU810 e um ESP32 DEVKIT V 1 a partir da comunicação serial assíncrona UART. A Figura 24 mostra a conexão feita entre os dispositivos, foram utilizados os pinos RF4 e RF5 da placa Easy PIC Fusion V7, onde o dsPIC está inserido, e que são respectivamente RX e TX, sendo a conexão realizada de forma cruzada nos pinos GPIO16 e GPIO17 que são RX2 e TX2 do ESP32. Como na Easy PIC Fusion V7 de desenvolvimento esses pinos estão ligados ao conector USB A, então o switch SW12, mais especificamente o 12.1 e 12.2 teve de ser alterado para o pino RF4 e RF5.

Figura 24 – Conexão entre os dispositivos.

Fonte: Elaborada pelo autor.

A primeira criptografia implementada no dsPIC foi a cifra Afim baseado nas funções matemáticas abordadas no capitulo 3. Foi criada uma função para cifrar *EncriptAfim()* e outra para decifrar *DecriptAfim()*, ambos recebendo os mesmos parâmetros sendo a mensagem a ser cifrado/decifrado, o tamanho do dado e as chaves *a* e *b*. Conforme visto na Figura 25, o laço de repetição for é utilizado para calcular os novos valores da mensagem.

Figura 25 – Bloco do código da cifra Afim.

```
int* EncriptAfim(char* vetor, int tamanho, int a, int b) {

    int msgC[tam];
    int i;

    for (i = 0; i < tam; i++) {
        msgC[i] = a * vetor[i] + b;
    }

    return msgC;
}

int* DecriptAfim(char* vetor, int tamanho, int a, int b) {

    int msgC[tam];
    int i;

    for (i = 0; i < tam; i++) {
        msgC[i] = (vetor[i] + b)/a;
    }

    return msgC;
}
```

Fonte: Elaborada pelo autor.

A cifra de Vigenère (Figura 26) foi escrita de modo semelhante à cifra Afim, sendo composta também por duas funções, a função *EncriptVigenere()* e *DecriptVigenere()* para cifrar e decifrar as mensagens, respectivamente. As funções recebem como parâmetros a mensagem, o tamanho da mensagem, a chave junto com o tamanho da chave.

Dentro do laço for o vetor que guarda a nova mensagem recebe a soma de cada caractere da mensagem original e dos caracteres da chave. Como a chave é menor que a mensagem, então sempre que o último caractere da chave é iterado, então a condição do *if* é verdadeira e então volta para o primeiro caractere da chave.

O algoritmo da cifra TEA, Figura 27, foi como já mencionado baseado no modelo proposto apresentado no artigo de David Wheeler e Roger Needham. Para guardar a informação referente ao procedimento para a cifragem, são utilizadas variáveis do tipo *uint32_t*, que nada mais é que um inteiro positivo de 32 bits. Desse modo, a mensagem e dividida e guardada em *V[0]* e *V[1*]. Dentro do laço for são realizadas as somas do mesmo modo que foi apresentado, onde *V0* vai receber as somas dos deslocamentos de 4 bits para esquerda usando "<<" de *V1*

Figura 26 – Bloco do código da cifra Vigenère.

```
int* EncriptVegenere(char* vetor, int tamanho, char* key, int tamanhoKey) {
    int msgC[tam];
    int aux = 0;
    int i;

    for (i = 0; i < tam; i++) {
        if (aux < tamanhoKey) {
            msgC[i] = vetor[i] + key[aux];
            aux++;
        }

        else {
            aux = 0;
        }

    }
    return msgC;
}

int* DecriptVegenere(char* vetor, int tamanho, char* key, int tamanhoKeyt) {
    int msgC[tam];
    int aux = 0;
    int i;

    for (i = 0; i < tam; i++) {
        if (aux < tamanhoKey) {
            msgC[i] = vetor[i] + key[aux];
            aux++;

        }

        else {
            aux = 0;
        }
    }
    return msgC;
}
```

Fonte: Elaborada pelo autor.

com a chave 0, *V1* deslocado 5 bits para direita usando “>>” e a soma de *V0* com a variável auxiliar sum que soma os valores de delta. Com auxílio do operador “^”, é feito o cálculo XOR e o resultado é somado a *V0*, e o mesmo ocorre para *V1*.

Como o processo para decifrar é realizar as operações invertidas, então na função *decrypt()* os valores de *V0* e *V1* são decrescidos utilizando as variáveis invertidas, conforme a Figura 27, sendo feitas as mesmas operações de deslocamento e XOR.

Figura 27 – Bloco do código da cifra TEA.

```
void encrypt(uint32_t* v) {
  uint32_t v0 = v[0], v1 = v[1], sum = 0, i;
  uint32_t delta = 0x9e3779b9;
  for (i = 0; i < 32; i++) {
    sum += delta;
    v0 += ((v1<<4) + KEY[0]) ^ (v1 + sum) ^ ((v1>>5) + KEY[1]);
    v1 += ((v0<<4) + KEY[2]) ^ (v0 + sum) ^ ((v0>>5) + KEY[3]);
  }

  v[0] = v0;
  v[1] = v1;
}

void decrypt (uint32_t* v) {
    uint32_t v0 = v[0], v1 = v[1], sum=0xC6EF3720, i;
    uint32_t delta=0x9e3779b9;

        for (i = 0; i < 32; i++) {
        v1 -= ((v0<<4) + KEY[2]) ^ (v0 + sum) ^ ((v0>>5) + KEY[3]);
        v0 -= ((v1<<4) + KEY[0]) ^ (v1 + sum) ^ ((v1>>5) + KEY[1]);
        sum -= delta;
    }

    v[0] = v0;
    v[1] = v1;
}
```

Fonte: Elaborada pelo autor.

A cifra RSA, por ser assimétrica necessita, de blocos de códigos para a criação dos pares de chaves. O código criado para esse projeto é baseado no código RSA escrito em C de Lanza (2018). Como já mencionado antes, a variável *e* deve ser maior que 1 e menor que *φ(n)* e, para encontrar um valor que satisfaça a condição, é utilizado um laço de repetição que varre todas as possibilidades entre 1 e *φ(n)* testando se o valor em questão atende às condições, conforme a Figura 28 onde é possível ver ainda a utilização da função *isPrimo()*, que testa se o valor em questão é ou não primo. Caso durante o laço *for* a variável *p* seja divisível por algum valor de 2 até a raiz quadrada de *p*, então a função retorna 0, caso não seja divisível por nenhum número retorna 1.

Figura 28 – Bloco do código para encontrar *e*.

```
for(i = 2; i <= phi - 1; i++){

        if(phi%i != 0 && verificaPrimo(i)){
                e = i;
                break;
        }
}
```

Fonte: Elaborada pelo autor.

A função vista na Figura 29 é utilizada para encontrar o valor *d*, tal que seja um inverso multiplicativo de *e*, para isso a função contém a implementação do algoritmo de Euclides. Dentro do comando *while*, até que a variável *r* seja igual a 0, deve ser realizado um conjunto de cálculos que consiste na divisão de *r* e *q*, afim de encontrar um resto r tal que seja maior ou igual a *b*. Porém, enquanto r ainda for maior que 0, as variáveis trocam de valor para que o processo continue até que seja encontrado o valor *d*. Caso o valor de *beta* seja negativo é somado a $\varphi(n)$, resultando assim nos números que integram os pares de chaves.

Figura 29 – Bloco do código MDC.

```
resto = phi;

while(resto != 0){

          if(e >= 0){
                q = 0;
                r = e;

                while(r >= phi){

                r -= phi;
                q++;
                }
          }

          e = phi;
          phi = resto;
          if(resto > 0){
                a2 = u - q *x;
                b2 = v - q * y;

                v = x;
                u = y;
                x = a2;
                y = b2;
          }
}

return b2;
```

Fonte: Elaborada pelo autor.

As próximas funções são para cifrar e decifrar as mensagens. Ambas as funções são vistas na Figura 30 e realizam um processo parecido com os anteriores, ou seja, modificando o

valor da mensagem original sendo que nesse caso é feita com a *mensagem* sendo elevada a *e mod n*, para cifrar, e elevado a *d mod n*, para decifrar.

Figura 30 – Função Potência.

```
int *encrypt(char *msg, long long e, long long n, int t){

        int i;
        int *msgC;

        mensagemC = malloc(t * sizeof(long long));
        for(i = 0; i < t; i++){
                msgC[i] = pow(msg[i], e, n);
        }

        return msgC;
}

char *encrypt(int *msgC, long long d, long long n){

        int i;
        char *mensagem;

        mensagem = malloc(t * sizeof(char));

        for(i = 0; i < t; i++){
                msg[i] = pow(msgC[i], d, n);
        }

        return msg;
}
```

Fonte: Elaborada pelo autor.

Para testar as cifras foi utilizado o display TFT cujo os códigos foram adaptados dos exemplos encontrados no próprio compilador, estão presentes na Figura 31. Basicamente as três funções principais são a função *ClearLbl()*, que serve para apagar o conteúdo escrito na *label*, mudando a cor da fonte para uma cor tal que a torne invisível no display, como por exemplo a cor branco, que é cor de fundo da tela. Em seguida, é realizada uma nova escrita copiando para o *label* um caractere vazio “ ”, ou seja um espaço em branco como é visto na função *ClearLabel()*. E a função *DrawLbl*, que seta as configurações referentes do label e então escreve nela a cadeia de caracteres desejada.

Figura 31 – Bloco de códigos das funções do display TFT.

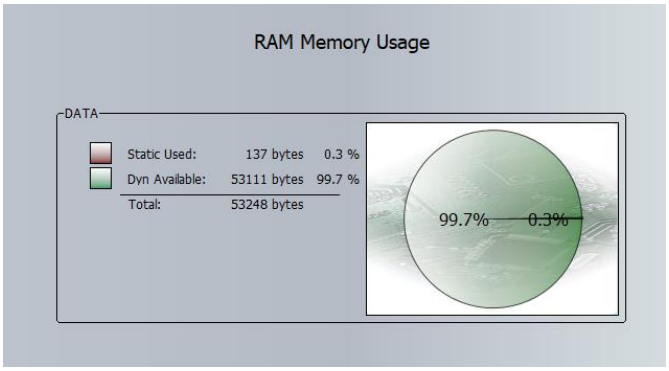
```
#include "AFIM_TFT_objects.h"
#include "AFIM_TFT_resources.h"

//---------------------- User code ----------------------//

//----------------- End of User code ------------------//

// Event Handlers

TLabel *label_selected = &label71;

void ClearLbl( TLabel *_label) {
  TFT_Set_Font(_label->FontName, /*CL_RED*/ Box1.Color /*CL_WHITE*/, FO_HORIZONTAL);
  TFT_Write_Text(_label->Caption, _label->Left, _label->Top);
}

void DrawLbl( TLabel *_label) {
  TFT_Set_Font(_label->FontName, _label->Font_Color, FO_HORIZONTAL);
  TFT_Write_Text(_label->Caption, _label->Left, _label->Top);
}

void ClearLabel( TLabel *_label) {
  ClearLbl(_label);
  strcpy(_label->Caption, " ");
}
```

Fonte: Elaborada pelo autor.

Foi feita também uma comunicação serial UART entre o dsPIC e um ESP32 para obter os dados referentes aos tempos gastos no processo, onde o dsPIC recebe uma mensagem do ESP32, cifra e envia de volta cifrada. A partir deste ambiente foi feita a implementação via Wi-Fi entre os dois ESP32 para que a mensagem chegue ao ESP32 alvo. A comunicação UART foi configurada de forma igual nos dois dispositivos para um *baud rate* de 9600 bits/s, sem paridade e com 1 bit de parada.

Os quatro testes propostos neste trabalho objetivam analisar o desempenho das cifras implementada. Para esse fim, foi escolhido analisar a quantidade de memória ROM e RAM utilizado, além da quantidade de tempo necessária para realizar os processos e o consumo de energia durante o período. Para obter a quantidade de memória ROM e RAM foi utilizada uma ferramenta disponibilizada pelo Mikro C. A ferramenta Statistics Figura 32 gera um relatório sobre as rotinas compiladas no dsPIC.

Figura 32 – Ferramenta Statistics do Mikro C for dsPIC.



Fonte: Elaborada pelo autor.

Para calcular o tempo gasto pelo dsPIC, foi feito a configuração, conforme a Figura 33, de um temporizador com uma frequência de 140 Mhz para contar o tempo em microssegundos já que, dada à complexidade das cifras e o poder de processamento do microcontrolador, é possível estimar que seja essa a faixa de tempo.

Figura 33 – Configuração do Timer 2.

```
void InitTimer2(){
  T2CON          = 0x8000;
  T2IE_bit          = 1;
  T2IF_bit          = 0;
  IPC0                 = IPC0 | 0x1000;
  PR2                 = 70;
}

void Timer2Interrupt() iv IVT_ADDR_T2INTERRUPT{
  T2IF_bit          = 0;

  if(flag == 0 ){
      t++;
  }

}
```

Fonte: Elaborada pelo autor.

A obtenção da energia gasta pelo dsPIC pode ser feita de diversas maneira sendo que boa parte delas está atrelada ao uso de sensores ou outros tipos de hardware, porém por falta destes dispositivos, o cálculo da energia consumida foi realizado a partir de uma equação teórica

visto em Patterson e Hennessy (2014) que relaciona a tensão de operação do processador *V*, a capacitância da pinagem *C*, a frequência do clock *f e* o tempo de operação *t* dado por:

$$E = V^2 * C * f * t \quad (7)$$

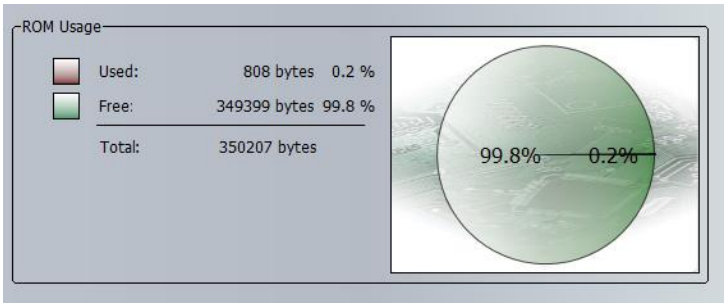
Como a frequência foi definida em 140 Mhz, com a tensão de operação de 3.3 V e capacitância dos pinos de 50 pF tirados da folha de dados do 33EP512MU810 tem-se:

$$E = 0{,}07623t \ Joule \quad (8)$$

5 RESULTADOS E DISCUSSÕES

Após os testes realizados com cada uma das cifras propostas para esse trabalho em termos dos parâmetros escolhidos, foi possível obter as quantidades de espaço de memória ocupado por cada cifra no dsPIC, desse modo foram escolhidas mensagens com valore de 72 bytes e chaves de 32 bytes, afim de equiparar ao tamanho da chave da cifra TEA implementada para esse trabalho. A Figura 34 demonstra a quantidade de memória ROM usada pela cifra Afim, onde é possível notar que o consumo não chega a 0.5% dos 350207 bytes disponíveis no microcontrolador.

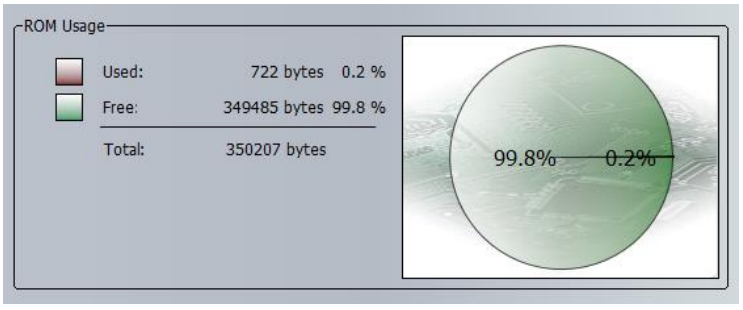
Figura 34 – Quantidade de memória ROM da cifra Afim.



Fonte: Elaborada pelo autor.

A cifra de Vigenère Figura 35, assim como a anterior, apresentou um valor abaixo dos 0.5% em relação a memória total, porém apresentando 86 bytes a menos que a cifra Afim.

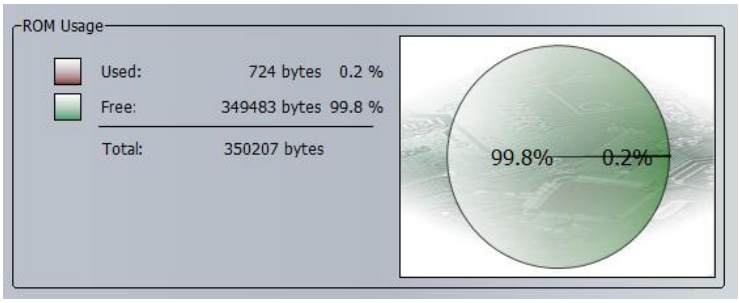
Figura 35 – Quantidade de memória ROM da cifra Vigenère.



Fonte: Elaborada pelo autor.

A Figura 36 mostra a quantidade de ROM para a cifra TEA, que teve seu valor mais uma vez próximo dos anteriores, ocupado 724 bytes.

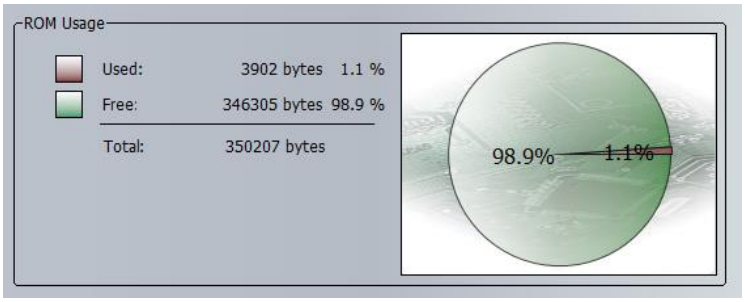
Figura 36 – Quantidade de memória ROM da cifra TEA.



Fonte: Elaborada pelo autor.

O último valor obtido de memória ROM é visto na Figura 37 e referente à cifra RSA, que dessa vez se distanciou dos outros valores, chegando a passar dos 1% de memória disponível ocupando 3902 bytes. Essa diferença é dada principalmente pelo fato de que por se tratar de uma cifra assimétrica, é necessário um conjunto de blocos de códigos para o cálculo das chaves necessárias.

Figura 37 – Quantidade de memória ROM da cifra RSA.

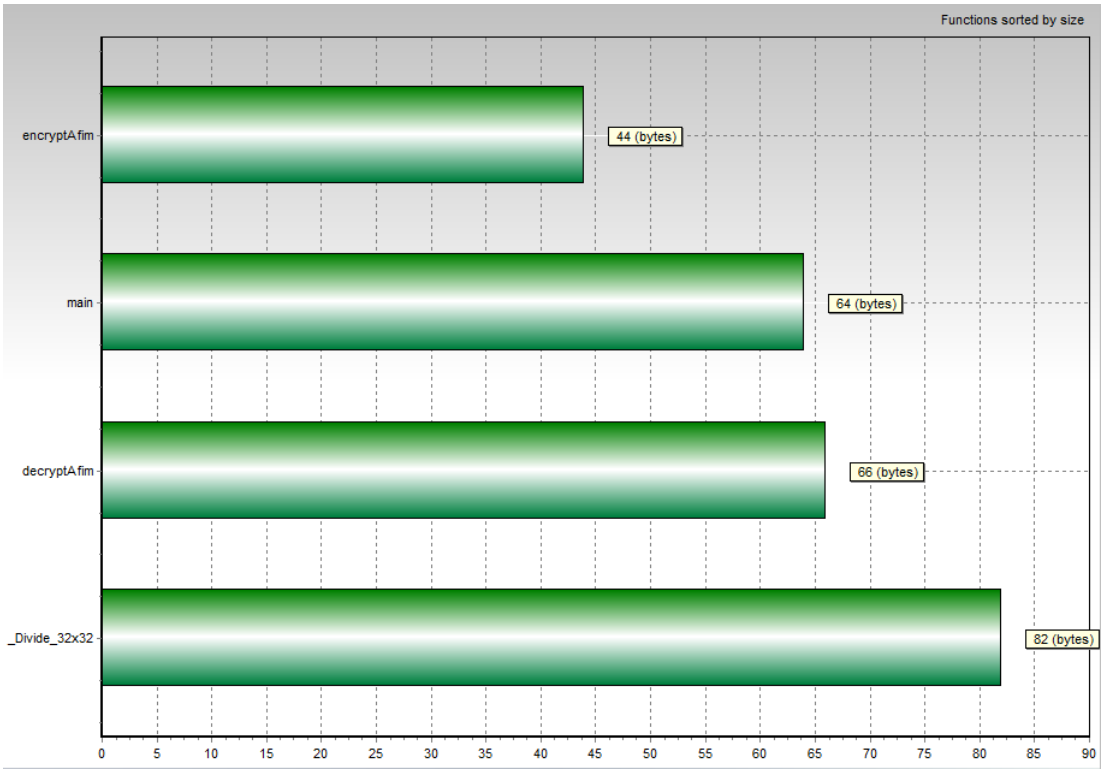


Fonte: Elaborada pelo autor.

Quanto ao uso de memória RAM utilizada por cada cifra, iniciando pela cifra Afim, a quantidade de memória utilizada pelas funções para cifrar e decifrar são vistas na Figura 38. A função *encryptAfim()* ocupou 44 bytes de memória, enquanto a função *decryptAfim()* ocupou 64 bytes. A diferença de memória utilizada para decifrar é maior devido a necessidade de realizar opções mais caras computacionalmente, como a divisão de números grandes que é possível ver na Figura 38 uma função *Divide_32x32()*, nativa do compilador utilizada para tal tarefa.

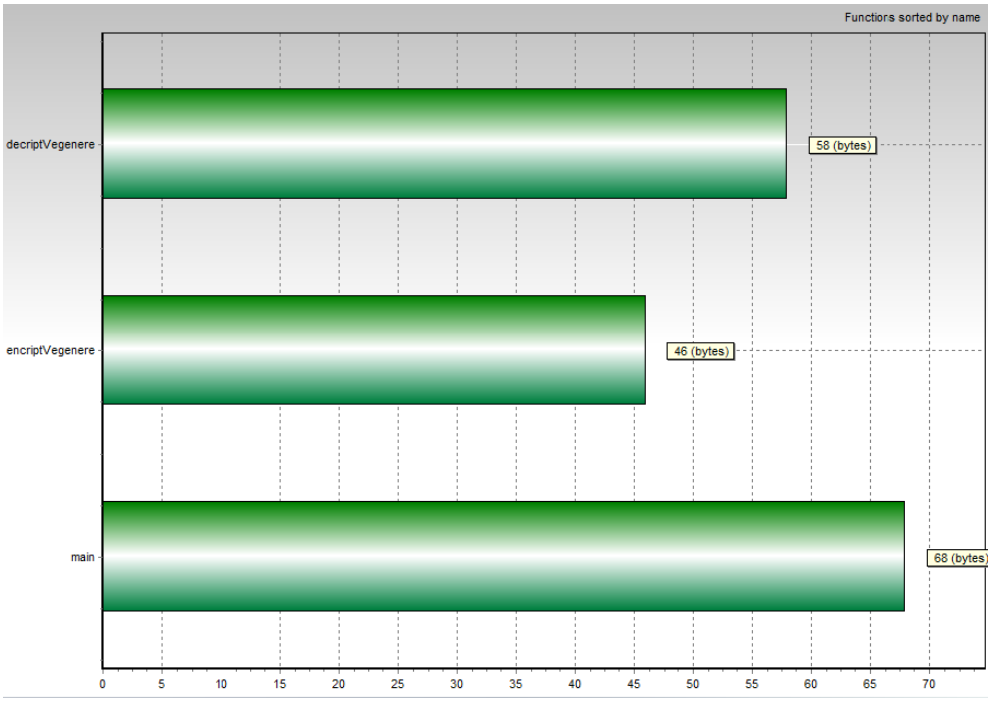
No caso da cifra de Vigenère conforme a Figura 39, as funções para cifrar *encriptVegenere()* e decifrar *decriptVegenere()* ocuparam, respectivamente, 46 bytes e 58 bytes, sendo valores próximos do encontrado para a cifra Afim.

Figura 38 – Quantidade de memória RAM da cifra Afim.



Fonte: Elaborada pelo autor

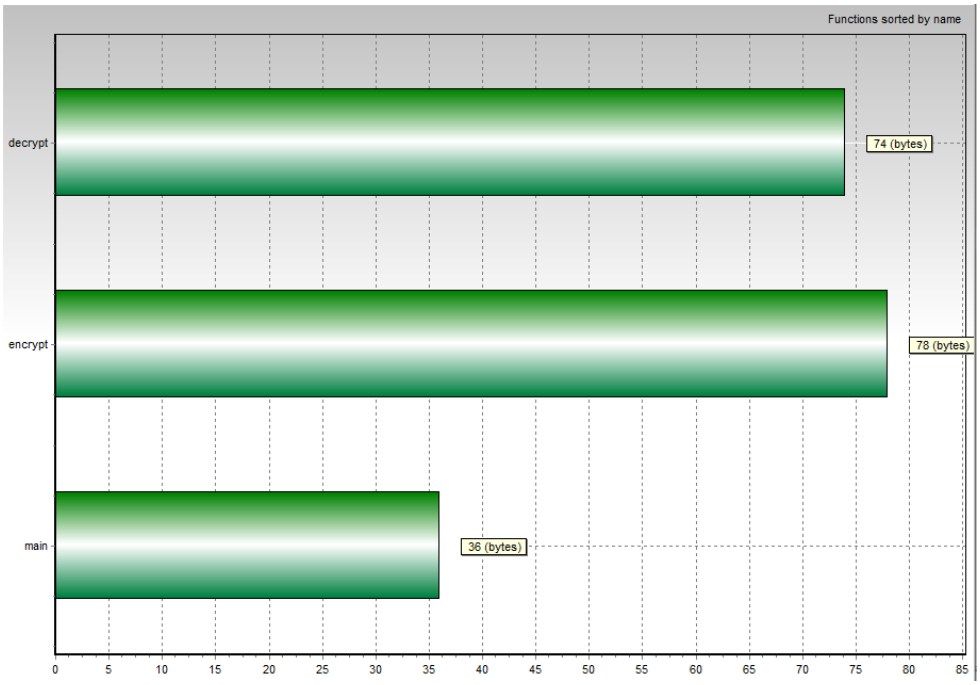
Figura 39 – Quantidade de memória RAM da cifra de Vigenère.



Fonte: Elaborada pelo autor.

Os valores de memória RAM para a cifra TEA tanto para cifrar, com a função *encrypt()*, como decifrar, com a função *decrypt()*, ocuparam respectivamente 78 e 74 bytes, conforme visto na Figura 40. Mais uma vez os valores ficaram próximos em relação aos obtidos anteriormente, mas com uma diferença menor entre as funções da cifra TEA, já que o procedimento é praticamente o mesmo tanto para encriptar como decriptar, ou seja, os cálculos realizados.

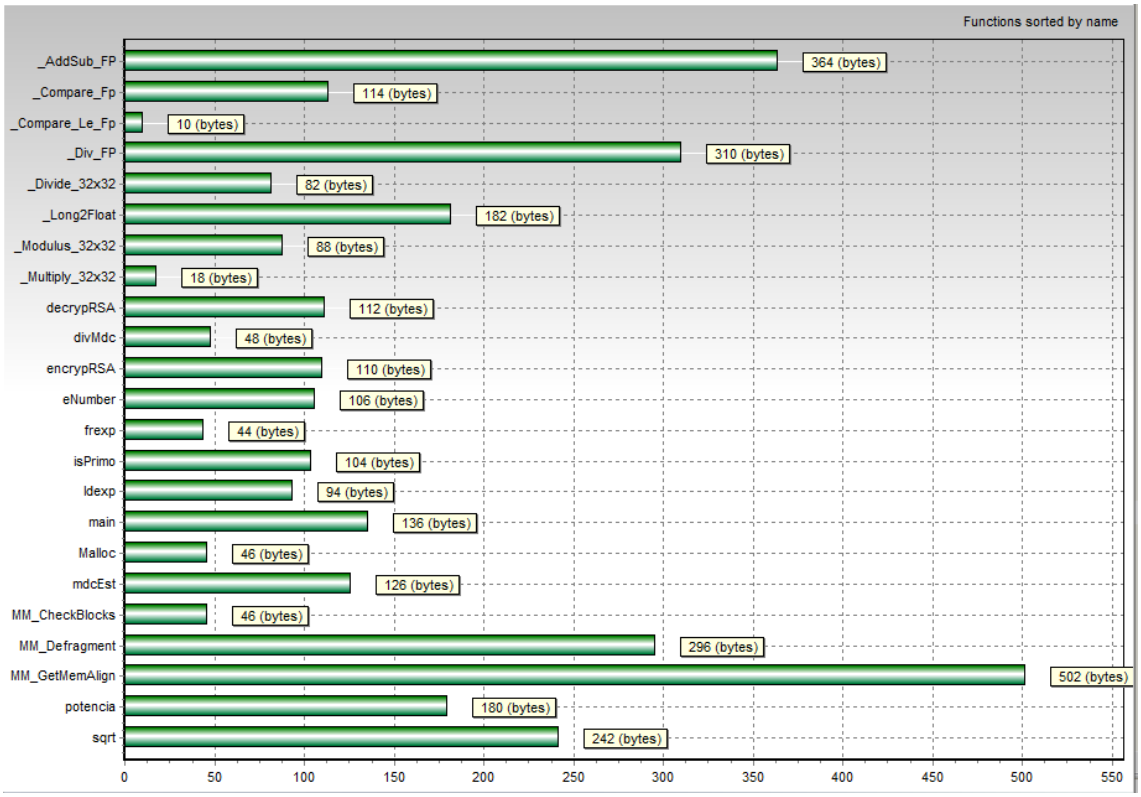
Figura 40 – Quantidade de memória RAM da cifra TEA.



Fonte: Elaborada pelo autor.

A cifra RSA, conforme mostra a Figura 41, conta com uma grande quantidade de funções nativas do compilador para realizar os cálculos necessários, tanto para gerar as chaves quanto o processo de encriptação e decriptação. Ainda conforme a Figura 41, as funções *encrypRSA()* e *decrypRSA()* consomem 110 e 112 bytes respectivamente.

Figura 41 – Quantidade de memória RAM da cifra RSA.



Fonte: Elaborada pelo autor.

O teste de performance das cifras para as quantidades de memória RAM e ROM utilizadas é visto de forma resumida na Tabela 3 onde, entre as cifras escolhidas, a que apresentou o maior consumo de memória ROM e RAM foi a RSA, e isso devido ao fato de ser uma cifra assimétrica e que necessita de códigos para a troca de chaves. As outras três cifras apresentam valores próximos de memória ROM, não ultrapassando dos 810 bytes.

Tabela 3 – Uso de memória RAM e ROM no dsPIC.

CIFRA	ROM	RAM (Cifrar)	RAM (Decifrar)
Afim	808 bytes	44 bytes	64 bytes
Vigenère	722 bytes	46 bytes	58 bytes
TEA	724 bytes	78 bytes	74 bytes
RSA	3902 bytes	110 bytes	112 bytes

Fonte: Elaborada pelo autor.

Quanto às funções para cifrar e decifrar, é observado que para as cifras de fluxo, como é o caso das cifras Afim, Vigenère e RSA, a quantidade de memória para decifrar é menor que

para cifrar, isso ocorre devido à necessidade de realizar procedimentos matemáticos que geram uma carga maior nos microcontroladores.

O tempo necessário para realizar a encriptação e decriptação foi realizado, para uma mensagem de 72 bytes e chaves de 32 bytes. O tempo necessário para cifrar e decifrar uma mensagem usando a cifra Afim ficou em 37µs para cifrar e 73µs para decifrar. Já Vigenère precisou de 33µs para cifrar e 42µs para decifrar uma mensagem conforme mostra na Tabela 4, onde é possível ver que os valores de tempo necessário são menores que na cifra Afim, principalmente quando para decifrar.

A cifra TEA levou 203µs para cifrar e 187µs para decifrar a mensagem segundo mostra a Tabela 4, os valores de tempo maiores que os apresentados pelas cifras Afim e Vigenère. Por fim, a cifra RSA apresentou valores de tempo para realizar o processo de gerar a chaves e cifrar em 211µs já para gerar a chave e decifrar ficou em 236µs, conforme visto nas Figuras 49 e 50, e de novo valores maiores que os anteriores, porém próximos da cifra TEA. Quanto às cifras RSA e TEA, pode-se notar que a TEA teve um desempenho melhor.

Tabela 4 – Tempo calculado para as cifras no dsPIC.

CIFRA	TEMPO (Cifrar)	TEMPO (Decifrar)
Afim	37µs	73µs
Vigenère	33µs	42µs
TEA	203µs	187µs
RSA	211µs	236µs

Fonte: Elaborada pelo autor.

A quantidade de energia consumida durante o processo, como já mencionado, foi realizada de forma teórica, assim para a obtenção dos valores foi feito o cálculo do tempo necessário não somente para cifrar/decifrar a mensagem, mas também para enviar de um ponto a outro. Deste modo, para fins de comparação, foi feito o processo de envio da mensagem original. Conforme a Tabela 4 é possível observar os valores estimados em relação ao tempo gasto no processo e, a partir da equação 2.0, é estimado o consumo de cada cifra com seus valores também vistos na Tabela 5.

Tabela 5 – Consumo de energia do dsPIC.

CIFRA	ENERGIA (Cifrar)	ENERGIA (Decifrar)
Afim	6,25086µJ	8,76645µJ
Vigenère	5,9459µJ	6,63201µJ
TEA	18,905µJ	17,6854µJ
RSA	19,5149µJ	21,4206µJ
Mensagem Original	3,43035µJ	3,43035µJ

Fonte: Elaborada pelo autor.

O tempo mensurado com o envio da mensagem teve um resultado idêntico ao visto na Tabela 4, onde teve apenas o acréscimo do tempo necessário para que as funções responsáveis pelo envio dos dados via serial fosse contabilizado. Quanto ao consumo de energia, as cifras Afim e de Vigenère mantiveram o consumo próximo ao da mensagem original demonstrando que as cifras não causam um impacto grande no consumo. Já as cifras TEA e RSA apresentaram um aumento no consumo de 551 % para TEA e 569 % para RSA, demonstrando que a cifra simétrica apresenta desempenho melhor que a assimétrica RSA.

6 CONCLUSÕES E PERSPECTIVAS

Com base nos resultados analisados, é possível concluir que todas as cifras escolhidas podem se enquadrar como cifras leves, uma vez que nenhuma delas ultrapassou quaisquer limites encontrados no microcontrolador dsPIC, pois em uma aplicação IOT, por exemplo, não chegaria a ocupar mais que 2% da memória disponível, em se tratando de memória ROM. Quanto à memória RAM utilizada pelas cifras é praticamente irrelevante.

Quanto ao tempo de operação, ou mesmo o consumo de energia, as cifras também apresentaram um desempenho satisfatório para sua capacidade de segurança, sendo que as cifras Afim e Vigenère se mostraram mais eficiente em termos computacionais, porém, devido sua fraqueza na análise de frequência, se tornam inúteis para aplicações, no entanto apresentam grande importância histórica e didática.

As cifras TEA e RSA, mesmo que apresentem uma eficiência menor que as anteriores, têm em seus algoritmos matemáticos a impossibilidade de serem quebradas utilizando técnicas como a análise em frequência, além de apresentarem desempenho digno de serem consideradas leves, mostrando o porquê de até hoje serem utilizadas.

Para possíveis análises futuras, seria interessante realizar a escolha de uma maior quantidade de cifras e que utilizem chaves de valores maiores acima dos 128 bytes, assim como também a implementação em uma aplicação IOT, afim de que seja avaliado quais as melhores possibilidades para a proteção dos dados compartilhados nesse meio.

7 REFERÊNCIAS

1. ALBARELLO, Rafael Hickmann. **Avaliação de algoritmos de criptografia e implementação de um protocolo leve para troca de chaves em dispositivos IOT**. 2019. 48 f. TCC (Graduação) - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Toledo, 2019.
2. ALMEIDA, Paulo J.. **Criptografia e Segurança**. 117 f. Universidade de Aveiro, Portugal, 2012.
3. ANJOS, Leandro Terra Versiani dos. **Segurança da Informação na Programação com uso de Criptografia e Certificação Digital**. 2016. 97 f. TCC - Curso de Tecnólogo em Sistemas de Computação, Universidade Federal Fluminense, Niterói, 2016.
4. ARDUINO. **Getting Started with Arduino products**. Disponível em: https://www.arduino.cc/en/Guide. Acesso em: 29 abr. 2021.
5. BONFIM, Daniele Helena. **Criptografia RSA**. 2017. 93 f. Dissertação (Mestrado) - Curso de Ciências, Instituto de Ciências Matemáticas e de Computação, São Carlos, 2017.
6. BORBA, Alex Alves; SILVA, Fernando Rodrigues da. **Elevador automatizado com comandos por voz**. 2016. 21 f. TCC (Graduação) - Curso de Tecnologia em Mecatrônica, Faculdade de Tecnologia de Garça, Garça, 2016.
7. CAP SISTEMA. **Referência de pinagem do ESP32**: quais pinos do gpio você deve usar?. Quais pinos do GPIO você deve usar?. 2020. Disponível em: https://capsistema.com.br/index.php/2020/03/22/referencia-de-pinagem-do-esp32-quais-pinos-do-gpio-voce-deve-usar/. Acesso em: 08 maio 2021.
8. CASTELLÓ, Thiago; VAZ, Verônica. **História da Criptografia**. Disponível em: https://www.gta.ufrj.br/grad/07_1/ass-dig/HistriadaCriptografia.html#:~:text=Hist%C3%B3ria%20da%20Criptografia&text=Desenvolvida%20por%20Arthur%20Scherbius%20em,ficaram%20conhecidas%20como%20Funkschl%C3%BCssel%20C. Acesso em: 07 maio 2021.
9. CASTRO, Felipe Lopes. **Criptografia RSA**: uma abordagem para professores do ensino básico. 2014. 61 f. TCC (Graduação) - Curso de Licenciatura em Matemática, Departamento de Matemática Pura, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2014.

10. ESPINDOLA, Marcelo. **Segurança em comércio eletrônico**. 2017. 14 f. Monografia (Especialização) - Curso de Gerência de Projetos de Tecnologia da Informação, Universidade do Sul de Santa Catarina, Santa Catarina, 2017.
11. Espressif Systems. Datasheet: ESP32. Electronic Publication, 2016.
12. GALDINO, Uelder Alves. **Teoria dos números e criptografia com aplicações básicas**. 2014. 77 f. Dissertação (Mestrado) - Curso de Matemática, Universidade Estadual da Paraíba, Campina Grande, 2014.
13. HUNN, Stephanie Ang Yee; NAZIRI, Siti Zarina Binti Md.; IDRIS, Norina Binti. The development of tiny encryption algorithm (TEA) crypto-core for mobile systems. In: 2012 IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS DESIGN, SYSTEMS AND APPLICATIONS (ICEDSA), 1., 2012, Arau. **2012 IEEE International Conference on Electronics Design, Systems and Applications (ICEDSA).** [S.L.]: Ieee, 2012. p. 45-49.
14. KASPERSKY DAILY. **Como uma cifra do sécula XVII se torna uma criptografia intransponível?** 2015. Disponível em: https://www.kaspersky.com.br/blog/vigenere-cipher-history/5688/. Acesso em: 02 maio 2021.
15. LEMOS JÚNIOR, Antonio Carlos. **Transmissor de Temperatura Multicanal com Comunicação RS485-MODBUS**. 2018. 148 f. Dissertação (Mestrado) - Curso de Inovação Tecnológica, Instituto de Ciências Tecnológicas e Exatas, Uberaba, 2018.
16. MATIC, Neboja. **EasyPIC Fusion V7**: User's Guide. MikroElektronika, 2013, 44 p.
17. MICROCHIP. **DsPIC33 Digital Signal Controllers for Advanced Motor Control**. Disponível em: https://www.microchip.com/en-us/solutions/motor-control-and-drive/motor-control-products/dspic33-digital-signal-controllers. Acesso em: 29 abr. 2021.
18. MICROCHIP. **DsPIC33EP512MU810**. Disponível em: https://www.microchip.com/wwwproducts/en/dsPIC33EP512MU810. Acesso em: 06 maio 2021.
19. Microchip. dsPIC33EPXXX, PIC24EPXXX. Electronic publication, 2011.
20. MIYADAIRA, Alberto Noboru. **Microcontroladores PIC 18**: aprenda e programe em linguagem c. São Paulo: Érica, 2012. 400 p
21. NAVARRO, Gabriela Matias. **Equalização de Frequência em Cifradores de Fluxo**: proposta de algoritmo. 2014. 123 f. TCC (Graduação) - Curso de Engenharia de Software, Universidade de Brasília, Brasília, 2014.

22. OLIVEIRA, Ronielton Rezende. Criptogradia simétrica e assimétrica: os principais algoritmos de cifragem. **Segurança Digital**, Niterói, v. 5, n. 1, p. 11-15, mar. 2012. Mensal.
23. PEREIRA, Nádia Marques Ikeda. **Criptografia**: uma nova proposta de ensino de matemática no ciclo básico. 2015. 78 f. Dissertação (Mestrado) - Curso de Mestrado Profissional em Matemática em Rede Nacional, Universidade Estadual Paulista, Ilha Solteira, 2015.
24. PONEMON INSTITUTE. **Global Encryption Trends Study**. Traverse City: Sem Editora, 2019. 8 p.
25. REVISTA GALILEU. **Segredos da máquina nazista Enigma são "quebrados" em exame de raios x**. Disponível em: https://revistagalileu.globo.com/Ciencia/noticia/2018/11/segredos-da-maquina-nazista-enigma-sao-quebrados-em-exame-de-raios-x.html. Acesso em: 29 abr. 2021.
26. ROBOCORE. **ESP32**: wifi & bluetooth. WiFi & Bluetooth. Disponível em: https://www.robocore.net/wifi/esp32-wifi-bluetooth. Acesso em: 30 abr. 2021.
27. SAKURAI, Rafael Guimarães. **Criptografia de chave privada**: simétrica. simétrica. 2020. Disponível em: http://www.universidadejava.com.br/outros/criptografia-simetrica/. Acesso em: 03 maio 2021.
28. SAKURAI, Rafael Guimarães. **Criptografia de chave pública ou assimétrica**. 2020. Disponível em: http://www.universidadejava.com.br/outros/criptografia-assimetrica/. Acesso em: 05 maio 2021.
29. SERAFIM, Vinícius da Silveira. **Introdução à criptografia**: cifras de fluxo e cifras de bloco. Rio Grande do Sul: Sem Editora, 2012. 4 p.
30. SHOEB, Mohammad; GUPTA, Vishal Kumar. A crypt analysis of the tiny encryption algorithm in key generation. **International Journal Of Communication And Computer Technologies.** India, p. 15-20. dez. 2012.
31. SILVA, Cristovam Lage da. **Desenvolvimento de um módulo de cripgrafia leve para FPGA utilizando o algoritmo Simon and Speck**. 2019. 36 f. TCC (Graduação) - Curso de Engenharia de Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2019.
32. SILVA, Willian Wallace de Matteus. **A evolução da criptografia e suas técnicas ao longo da história**. 2019. 29 f. TCC (Graduação) - Curso de Sistemas de Informação, Instituro Federal Goiano, Ceres, 2019.

33. TOSHIHIKO, Okamura. **Lightweight Cryptography Applicable to Various IoT Devices**. Disponível em: https://dr.nec.com.onenec.net/en/global/techrep/journal/g17/n01/170114.html?nid=gihoEN006. Acesso em: 30 abr. 2021.