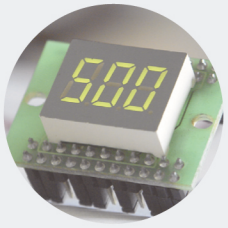
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






ALGORITMOS CRIPTOGRÁFICOS E O SEU DESEMPENHO NO ARDUINO

NUNO JOSE TEIXEIRA REIS BARBOSA
julho de 2017

POLITÉCNICO
DO PORTO

ALGORITMOS CRIPTOGRÁFICOS E O SEU DESEMPENHO NO ARDUINO

Nuno José Teixeira Reis Barbosa

Departamento de Engenharia Electrotécnica

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização em Telecomunicações

2017

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Nuno José Teixeira Reis Barbosa, Nº 1130203, 1130203@isep.ipp.pt

Orientação: Paula Maria Marques Moura Gomes Viana, pmv@isep.ipp.pt

Departamento de Engenharia Electrotécnica

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização em Telecomunicações

2017

Resumo

O Arduino é uma plataforma muito robusta e multifacetada utilizada em diversas situações e, cada vez mais, um elemento relevante na arquitetura da Internet das Coisas. Ao disponibilizar várias interfaces de comunicação sem fios, pode ser utilizado para controlar eletrodomésticos, portas, sensores de temperatura, etc. permitindo implementar facilmente a comunicação entre estas "coisas".

Nesta tese foram estudadas as principais redes sem fios utilizadas pelo Arduino (*Bluetooth Low Energy* [BLE], Wi-Fi e ZigBee) para tentar perceber qual a que tem o melhor desempenho, vantagens e desvantagens de cada uma, quais os módulos necessários para permitir ao Arduino utilizar esse tipo de rede sem fios, quais as principais funções para que foram projetadas quando criadas e qual o sistema de segurança utilizado nestas redes.

Estas diferentes tecnologias sem fios permitem uma maior mobilidade e uma maior flexibilidade no desenho das estruturas de rede do que as redes com fios convencionais. Porém, este tipo de redes têm uma grande desvantagem já que qualquer um dentro do alcance da rede sem fios consegue intercetar o sinal que está a ser transmitido. Para solucionar e proteger a informação que é transmitida por estas redes foram desenvolvidos vários algoritmos de criptografia. Estes dados encriptados só podem ser lidos por dispositivos que tenham uma determinada chave. Os algoritmos de criptografia *Data Encryption Standard* (DES), *Triple DES* (TDES), *Advanced Encryption Standard* (AES), *eXtended TEA* (XTEA) e *Corrected Block TEA* (XXTEA) estão entre as técnicas mais conhecidos e usadas atualmente. Nesta tese foram analisados estes algoritmos e as suas vulnerabilidades, tendo também sido feito um levantamento dos principais ataques existentes para avaliar se ainda são seguros atualmente.

De forma a avaliar a possibilidade de utilizar o Arduino em aplicações que utilizem comunicações sem fios com segurança, foram realizados testes de desempenho com os algoritmos de criptografia estudados, usando bibliotecas já existentes. Nos testes de desempenho implementados verificou-se que o AES é bastante mais rápido do que as outras

soluções, oferecendo ainda uma maior segurança. Já o TDES verificou-se ser bastante lento, justificando o porquê de o algoritmo ser pouco usado, sendo ao longo dos anos substituído pelo AES. O XXTEA ficou em posição intermédia no teste de desempenho, tendo uma relação segurança/desempenho interessante e revelando-se assim uma escolha melhor do que o TDES.

Palavras-Chave

AES, Arduino, Bluetooth, Criptografia, DES, Desencriptação, Encriptação, Internet das Coisas, segurança, TDES, TEA, Wi-Fi, XTEA, XXTEA, ZigBee.

Abstract

The Arduino is a very robust and multifaceted platform used in many situations and, increasingly, a relevant element in the Internet of Things. By providing several wireless communication interfaces, it can be used to control household appliances, doors, temperature sensors, etc. Allowing easy implementation of communication between these "things".

In this thesis the main wireless networks used by Arduino (Bluetooth Low Energy [BLE], Wi-Fi and ZigBee) were studied to try to understand which one has the best performance, the advantages and disadvantages of each one, the modules needed to implement each wireless network and what security system are used.

These different wireless technologies allow for greater mobility and greater flexibility in the design of network structures than conventional wired networks. However, such networks have a major disadvantage since anyone within the range of the wireless network can intercept the signal being transmitted. Several cryptographic algorithms have been developed to solve and protect the information that is transmitted by these networks. This encrypted data can only be read by devices that have a certain key. Triple Encryption Standard (DES), Advanced Encryption Standard (AES), eXtended TEA (XTEA) and Corrected Block TEA (XXTEA) encryption algorithms are among the best known and currently used algorithms. In this thesis these algorithms have been analyzed to compare their vulnerabilities and to identify the main existing attacks.

In order to evaluate the possibility of using Arduino in applications that use wireless communications with security, performance tests were implemented using existing libraries. The results show that the AES is much faster than the other algorithms, offering even greater security. TDES was found to be quite slow, justifying why the algorithm has little used, and why over the years has been replaced by AES. The XXTEA was ranked in the middle of the performance test, having an interesting safety/performance ratio proving it to be a better choice than TDES.

Keywords

AES, Arduino, Bluetooth, DES, Decryption, Encryption, Internet of Things, security, TDES, TEA, Wi-Fi, XTEA, XXTEA, ZigBee.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AES	–	Advanced Encryption Standard
ATT	–	Attribute Protocol
BLE	–	Bluetooth Low Energy
CBC-MAC	–	Cipher Block Chaining-Message Authentication Code
DES	–	Data Encryption Standard
DSSS	–	Direct Sequence Spread Spectrum
EEPROM	–	Electrically Erasable Programmable Read-Only Memory
EFF	–	Electronic Frontier Foundation
FEAL	–	Fast data Encipherment ALgorithm
FPGA	–	Field Programmable Gate Array
GCC	–	GNU Compiler Collection
HTTPS	–	Hyper Text Transfer Protocol Secure
I2C	–	Inter-Integrated Circuit
IBM	–	International Business Machines
ICSP	–	In-Circuit Serial Programming
IEEE	–	Institute of Electrical and Electronics Engineers
ISM	–	Industrial, Scientific and Medical
LCD	–	Liquid Crystal Display
MIMO	–	Multiple-Input Multiple-Output

MISO – Master In Slave Out

MOSI – Master Out Slave In

MU-MUMO – Multi-User MIMO

NIST – National Institute of Standards and Technology

OFDM – Orthogonal Frequency Division Multiplexing

PIC – Peripheral Interface Controller

PWM – Pulse-Width Modulation

RISC – Reduced Instruction Set Computing

RX – receive

SCK – Serial Clock

SCL – Serial Clock

SDA – Serial Data

SIM – Sieve-In-the-Middle

SMD – Surface-Mount Device

SoC – System-on-a-Chip

SPARC – Scalable Processor ARChitecture

SPI – Serial Peripheral Interface

SPN – Substitution-Permutation Network

SRAM – Static Random Access Memory

SS – Slave Select

TDES – Triple Data Encryption Standard

TEA – Tiny Encryption Algorithm

TX – Transmit

WPA – Wi-Fi Protected Access

XOR – eXclusive OR

XTEA – eXtended TEA

XXTEA – Corrected Block TEA

1. INTRODUÇÃO

Historicamente, a criptografia surgiu como meio para permitir a troca de informações, mantendo a privacidade, mesmo na presença de um adversário com acesso ao canal de comunicação. Embora a privacidade continue a ser um objetivo central, com o surgimento dos computadores o campo de aplicação expandiu-se para abranger outros objetivos, como garantir a integridade e autenticidade das comunicações.

Com o aparecimento das redes sem fios, a privacidade tornou-se ainda mais importante, pois foi necessário garantir a encriptação da informação trocada pelo ar. Sem ela, um atacante poderia ter acesso às informações enviadas por uma determinada rede, bastando para isso estar no alcance dessa mesma rede. Para proteger as informações que são enviadas tanto nas redes sem fios como com fios, foram desenvolvidos ao longo dos anos vários algoritmos de criptografia que utilizam fórmulas matemáticas complexas para encriptar todas as informações que o emissor quer proteger. Estas informações só podem ser lidas por alguém ou por algum dispositivo que tenha uma chave para conseguir descodificar o conteúdo enviado.

Atualmente, este problema avança para outro patamar com o surgimento da Internet das Coisas ou *Internet of Things* (LoT), em que se tem uma grande parte das "coisas" que se usa diariamente ligadas à Internet a comunicarem entre si, desde eletrodomésticos, meios de transporte, *wearables*, dispositivos médicos, *smart tvs*, etc. Esta evolução não seria possível sem a criação e o melhoramento das redes sem fios existentes. Contudo, também é necessário que haja privacidade, segurança e autenticidade neste tipo de comunicações, tornando os algoritmos de criptografia muito importantes, tanto em desempenho como em segurança. Estes algoritmos têm que ser leves, já que muitos dos dispositivos têm uma capacidade de processamento reduzida e muitos deles funcionam a bateria.

O Arduino, sendo uma placa eletrónica pequena com uma plataforma de programação muito simples e cheio de recursos e acessórios, é uma excelente escolha para ser utilizada na

Internet das Coisas. Com ele é possível fazer de quase tudo um pouco, desde enviar informações para a nuvem, verificar os sensores de temperatura, operar atuadores, analisar qual o canal favorito do utilizador, a que horas o utilizador chega a casa, trancar automaticamente as portas, acionar o alarme, desligar as luzes, monitorizar os consumos energéticos, etc.

Existem já muitas bibliotecas criadas para a plataforma Arduino, onde estão implementadas as funções e os procedimentos necessários para realizar este tipo de comunicações. Também existem bibliotecas desenhadas com as funções matemáticas e os procedimentos dos mais conhecidos algoritmos de criptografia para ser possível encriptar os dados que são enviados por esta placa.

1.1. OBJETIVOS

Os principais objetivos desta tese são:

- Fazer um estudo das melhores e mais utilizadas técnicas de encriptação para o Arduino (DES, TDES, AES, XTEA e XXTEA);
- Estudar as principais vulnerabilidades de cada um dos algoritmos;
- Fazer um levantamento dos ataques existentes para os algoritmos, de modo a perceber se e o quanto são seguros;
- Realizar testes de desempenho dos diferentes algoritmos de criptografia estudados no Arduino, utilizando bibliotecas já criadas com as funções matemáticas destes algoritmos, para perceber quais são os mais rápidos a realizar a encriptação e a desencriptação tendo em conta a quantidade de dados de entrada;
- Estudar as principais redes sem fios utilizadas no Arduino (WiFi, *Bluetooth Low Energy* [BLE] e ZigBee), de modo a analisar as vantagens e desvantagens de cada uma, perceber para que aplicações elas foram criadas e escolher os melhores módulos para serem integrados no Arduino e utilizados em projetos com este tipo de tecnologia sem fios;

- Estudar os sistemas de segurança utilizados pelas principais redes sem fios do Arduino.

1.2. ORGANIZAÇÃO DO RELATÓRIO

Nesta tese começa por fazer-se uma pequena introdução à criptografia e à sua história. Depois é feito um estudo sobre as principais cifras de bloco e o levantamento dos ataques mais eficazes e conhecidos contra estas cifras. Isto tudo é apresentado no capítulo 2.

No capítulo 3 é feito um estudo de desempenho das cifras identificadas no capítulo 2, usando um Arduino Mega 2560 e as bibliotecas já criadas com estes algoritmos criptográficos.

No capítulo 4 são analisadas as vantagens e desvantagens dos principais protocolos de ligação de dados sem fios utilizados para ligar o Arduino a atuadores e sensores, de modo a perceber qual a ligação de dados mais indicada para ser usada tendo em conta a aplicação final. Neste capítulo também são analisados os sistemas de segurança utilizados por estes protocolos.

No capítulo final são apresentadas as principais conclusões de todo este estudo realizado ao longo da tese. Em anexo são incluídos os programas usados para fazer o teste de desempenho dos algoritmos criptográficos no Arduino.

2. CRIPTOGRAFIA

A criptografia é uma ciência que estuda diferentes métodos para enviar uma determinada informação de uma forma segura. Essa informação só poderá ser lida por alguém que tenha uma determinada chave, especificada na encriptação. Essa chave pode ser comparada a uma chave vulgar para abrir e fechar portas, em que a fechadura de uma porta tem uma estrutura específica de pinos com múltiplas posições possíveis – se as posições ditadas pela chave corresponderem à posição específica dos pinos da fechadura da porta, a porta abre, senão não. As chaves de encriptação podem ser de dois tipos: chaves simétricas (em que o emissor e o recetor partilham a mesma chave) e chaves assimétricas (em que são usadas duas chaves diferentes, mas matematicamente relacionadas).

A criptografia consiste em dois processos: encriptação e desencriptação. A encriptação pega numa mensagem original que se queira enviar (texto puro) e usa um determinado algoritmo (cifra) na mensagem original, de modo a transformá-la num texto ilegível (texto cifrado). A desencriptação é o inverso, ou seja, recebe um texto ilegível e usa um algoritmo para transformá-lo de volta na mensagem original.

A criptografia tem como principais objetivos os seguintes aspetos:

- **Autenticação:** o processo de provar a identidade;
- **Privacidade/confidencialidade:** a garantia que ninguém pode ler a mensagem, exceto o destinatário;
- **Integridade:** assegurando ao recetor que a mensagem original não foi alterada de alguma maneira até chegar a ele;
- **Não-rejeição:** um método para provar que o remetente realmente enviou a mensagem.

2.1. HISTÓRIA

A criptografia começou por ser usada há milhares de anos e usava métodos simples com papel e caneta ou alguma mecânica simples. Essa criptografia é conhecida como criptografia clássica. O primeiro uso conhecido da criptografia remonta em uma inscrição esculpida por volta de 1900 aC, na câmara principal do túmulo do nobre *Khnumhotep II*, no Egito [65]. Apesar de essa inscrição não ser uma forma de escrita secreta, ela incorpora algum tipo de transformação do texto puro. Outras evidências de usos de criptografias podem ser encontradas nas grandes civilizações antigas.

Mais tarde, em 100 aC aproximadamente, Júlio César era conhecido por usar uma forma de criptografia para comunicar com os seus generais. Essa cifra era um tipo de cifra de substituição conhecida como cifra de César. Numa cifra de substituição cada letra do texto puro é substituída por outra, originando o texto cifrado. A variante usada por César era uma mudança por 3, ou seja, cada letra era substituída por outra, 3 posições mais à frente do alfabeto: a letra ‘A’ era substituída por ‘D’, a letra ‘B’ substituída por ‘E’ e assim sucessivamente. No final do alfabeto, essa contagem até 3 voltava ao início do alfabeto, por exemplo, o ‘X’ era substituído por ‘A’. Na figura 1 está ilustrado um esquema onde se exemplifica o funcionamento desta cifra. Neste tipo de cifras é necessário esconder o sistema que é utilizado, senão é fácil descodificar a mensagem encriptada, pois não utiliza nenhuma chave de encriptação. No entanto, as cifras de substituição podem ser facilmente quebradas com a análise das frequências das letras, sabendo a língua utilizada na comunicação.

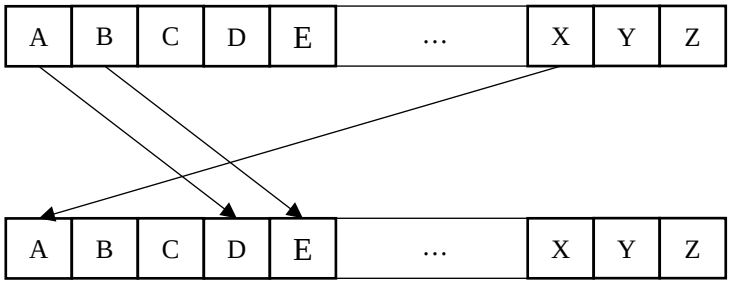


Figura 1 Cifra de César

Posteriormente, no século XVI, *Vigenère* criou uma cifra que ficou conhecida por ser a primeira a usar uma chave de encriptação. A chave de encriptação era repetida várias vezes,

de modo a ficar com o mesmo tamanho da mensagem que se queria enviar. Depois o texto cifrado era produzido adicionando com módulo 26 a letra da mensagem com a letra da chave. Contudo, a cifra de *Vigenère*, tal como a cifra de César, também pode ser facilmente quebrada, porque como a chave de encriptação é repetida várias vezes, as palavras comuns como "de" vão aparecer encriptadas segundo as mesmas letras, levando à descoberta de padrões repetidos no texto cifrado. Então pode-se dizer que a cifra de *Vigenère* depende da chave de encriptação e não do sistema. Na tabela 1 está representado um exemplo de uma encriptação usando a cifra de *Vigenère* para encriptar o texto "VERNAMCIPHER".

Tabela 1 Encriptação usando a cifra de *Vigenère*

Texto puro	V	E	R	N	A	M	C	I	P	H	E	R
Número equivalente	21	4	17	13	0	12	2	8	15	7	4	17
+ Número aleatório	76	48	16	82	44	3	58	11	60	5	48	88
= Sum	97	52	33	95	44	15	60	19	75	12	52	105
= mod 26	19	0	7	17	18	15	8	19	23	12	0	1
Texto cifrado	t	a	h	r	s	p	i	t	x	m	a	b

No início do século XIX, *Hebern* criou uma máquina eletromecânica que usava um rotor, sendo a chave secreta incorporada num disco rotativo. A chave codificava uma tabela de substituição e, quando se escrevia no teclado, resultava um texto cifrado na saída. Esta máquina também era quebrável usando a análise de frequência das letras.

No final da 1ª Guerra Mundial, o engenheiro alemão *Arthur Scherbius* inventou a máquina *Enigma*. Esta máquina foi muito utilizada pelos alemães durante a 2ª Guerra Mundial. A máquina usava 3, 4 ou mais rotores. Estes rotores rodavam a velocidades diferentes, dependendo da velocidade que se escrevia no teclado. Esta máquina foi quebrada pela Polónia e, mais tarde, os britânicos arranjaram forma de quebrar a chave utilizada pela máquina num dia. Na figura 2 é mostrada uma foto desta máquina.

Figura 2 Máquina Enigma [1]

Até ao fim da 2ª Guerra Mundial, a criptografia era maioritariamente utilizada para fins militares. Foi a partir daqui que a criptografia começou a ganhar atenção pelas empresas para protegerem as suas informações contra olhares indiscretos, principalmente da concorrência. Foi a partir do inicio da década de 70 que nasceu a criptografia moderna com o desenvolvimento da cifra *Lucifer* pela *International Business Machines* (IBM), que mais tarde se tornou o padrão nacional dos EUA e mudou de nome para *Data Encryption Standard* (DES).

2.2. *DATA ENCRYPTION STANDARD* (DES)

Em 1973, o *National Institute of Standards and Technology* (NIST) publicou um pedido de propostas para sistemas de criptografia de chave simétrica para ser utilizado como padrão a nível nacional nos EUA. A proposta da IBM com a cifra *Lucifer* foi a escolhida, ficando no final com o nome *Data Encryption Standard* (DES). O DES foi publicado no *Federal Information Processing Standards* (FIPS) em 1975. Após a publicação, o DES foi muito criticado pelo pequeno tamanho da chave (só 56 bits), o que poderia tornar vulnerável a ataques do tipo força bruta. Mesmo assim, depois da sua publicação a cifra foi uma das cifras de bloco de chave simétrica mais utilizadas.

O DES é uma cifra de bloco, ou seja, realiza a encriptação do texto puro em blocos (neste caso cada bloco tem 64 bits), gerando blocos de texto cifrado, normalmente do mesmo tamanho. A escolha do tamanho dos blocos não afeta a segurança das cifras. A força da cifra depende do tamanho da chave de encriptação. Na figura 3 pode ver-se um pequeno esquema que explica a cifra DES de uma maneira simplificada.

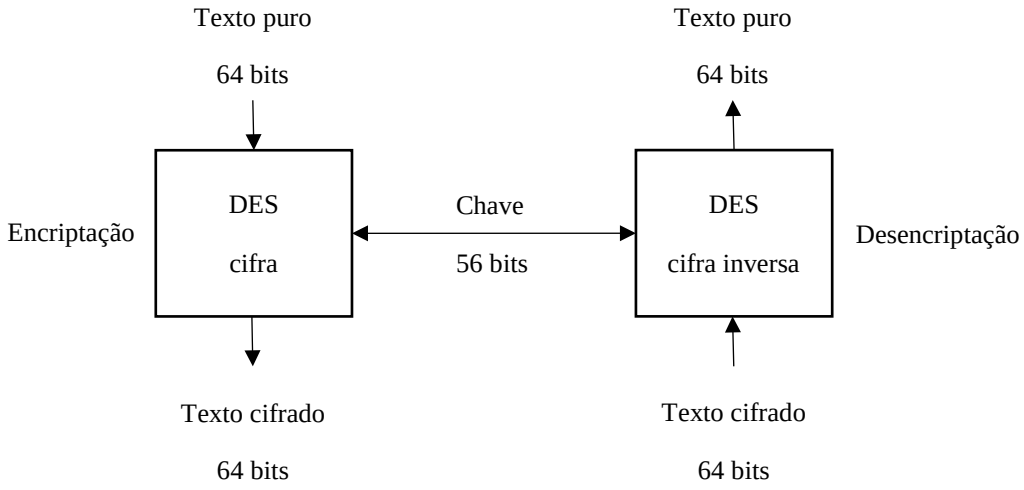


Figura 3 Estrutura geral do DES

O processo de encriptação do DES utiliza duas permutações (*P-boxes*), conhecidas como permutação inicial e final, mais dezasseis rondas de *Feistel* (estrutura simétrica usada na construção das cifras de bloco). Cada ronda usa uma chave de ronda de 48 bits diferente, gerada a partir da chave original da cifra. Na figura 4 está representado o processo de encriptação do DES.

As permutações inicial e final são inversas uma da outra. Cada uma das permutações permuta os 64 bits de entrada de acordo com uma regra predefinida ilustrada na tabela 2. Cada lado da tabela pode ser pensado como uma matriz de 64 elementos. O valor de cada elemento define o número da porta de entrada e a ordem do elemento define o número de porta da saída.

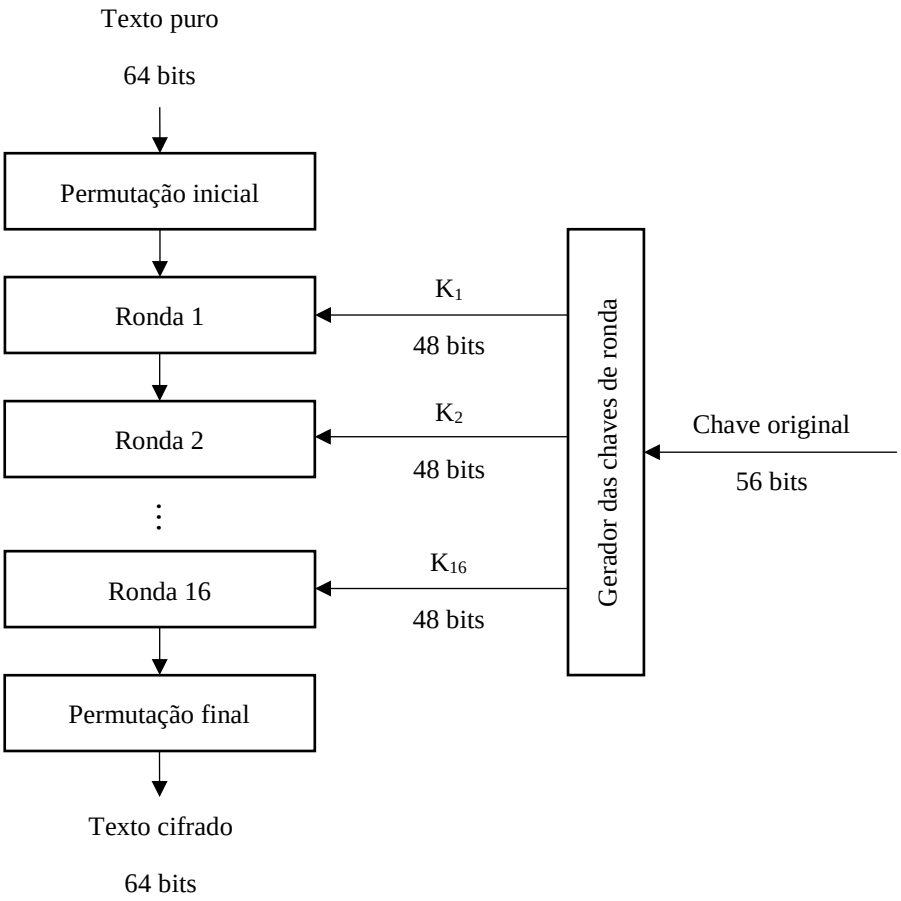


Figura 4 Encriptação do DES

Tabela 2 Regra das permutações

Permutação inicial								**Permutação final**							
58	50	42	34	26	18	10	02	40	08	48	16	56	24	64	32
60	52	44	36	28	20	12	04	39	07	47	15	55	23	63	31
62	54	46	38	30	22	14	06	38	06	46	14	54	22	62	30
64	56	48	40	32	24	16	08	37	05	45	13	53	21	61	29
57	49	41	33	25	17	09	01	36	04	44	12	52	20	60	28
59	51	43	35	27	19	11	03	35	03	43	11	51	19	59	27
61	53	45	37	29	21	13	05	34	02	42	10	50	18	58	26
63	55	47	39	31	23	15	07	33	01	41	09	49	17	57	25

Estas duas permutações não têm nenhuma importância na criptografia do DES, pois as permutações são predefinidas e não utilizam nenhuma chave. Por isso, não se sabe ao certo

a razão da implementação destas permutações. Suspeita-se que é para impedir a simulação em *software*, fazendo com que a cifra só funcione em *hardware*.

O DES tem 16 rondas, em que cada ronda pega no L_{I-1} e R_{I-1} da ronda anterior ou da permutação inicial (se for a ronda 1) e cria o L_I e o R_I que envia para a próxima ronda ou permutação final (se for a ronda 16). A ronda cria o L_I e o R_I através de dois elementos: o misturador e o *swapper*. Ambos são invertíveis, o *swapper* só troca o texto do lado esquerdo para o lado direito e o misturador é invertível porque só usa a operação *eXclusive OR* (XOR). Todos os elementos que não são invertíveis ficam guardados dentro da função $f(R_{I-1},K_I)$. Na figura 5 está representada uma ronda comum do DES.

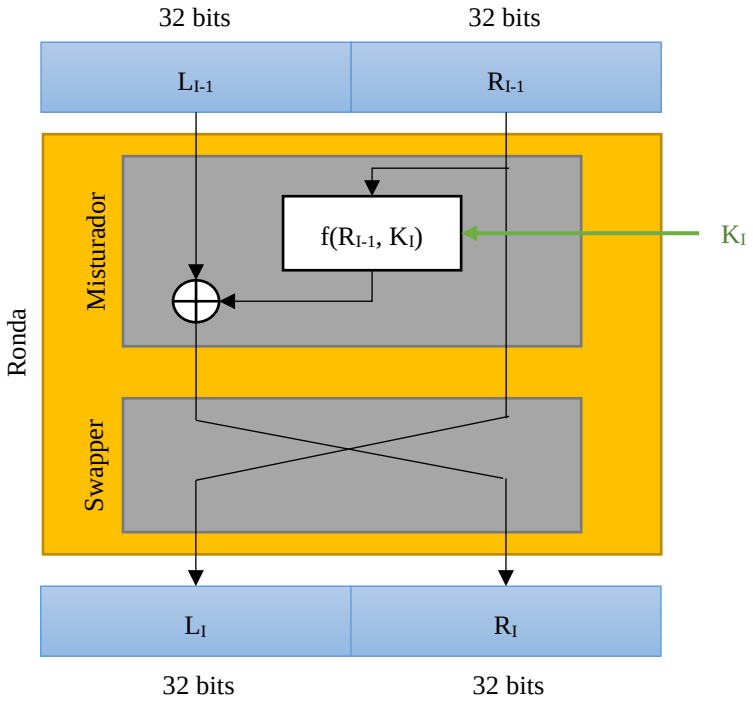


Figura 5 Uma ronda do DES

A função $f(R_{I-1}, K_I)$ que está em cada ronda do DES é a função principal desta cifra. Esta função pega na chave de 48 bits, mais os 32 bits mais à direita da ronda anterior ou permutação inicial (se for a ronda 1) e produz uma saída de 32 bits. Na figura 6 pode ver-se mais pormenorizadamente como é constituída esta função.

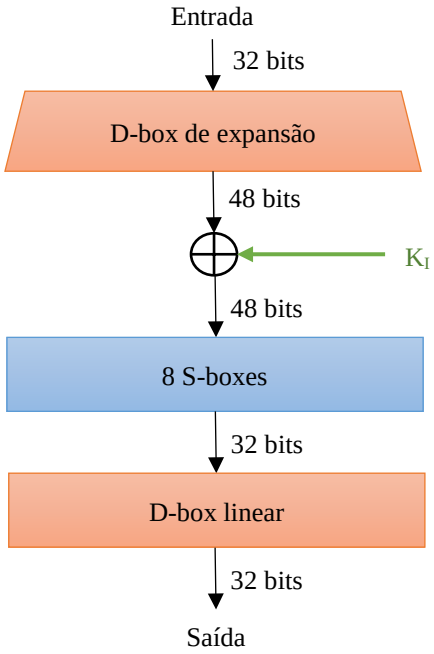


Figura 6 Função DES

A função DES primeiro precisa de expandir a entrada (R_{I-1}) de 32 bits para 48 bits, para depois ser possível realizar a operação XOR com a chave de ronda (K_I) de 48 bits. Este passo é realizado pela *D-box* de expansão. Ela divide em 8 seções os bits de entrada (R_{I-1}) com 4 bits cada uma. Cada seção de 4 bits é expandida para 6 bits através de uma regra predeterminada: os bits de entrada 1, 2, 3 e 4 são copiados para os bits de saída 2, 3, 4 e 5, respetivamente, o bit 1 da saída vem do bit 4 da seção anterior e o bit 6 vem do bit 1 da próxima seção. Como as seções 1 e 8 são consideradas seções adjacentes, então o mesmo aplica-se aos bits 1 e 32.

Depois da operação XOR, a função DES tem 8 caixas de substituição (*S-boxes*), em que cada uma tem uma entrada de 6 bits e uma saída de 4 bits. Estas *S-boxes* têm como objetivo misturar os bits. Os 48 bits que chegam da operação XOR são divididos em 8 blocos de 6 bits e cada um desses blocos vai para uma *S-box*. A saída de 4 bits de cada *S-box* resultará um total de 32 bits de saída. A substituição que cada *S-box* realiza obedece a uma regra predefinida com base numa tabela de 4 linhas e 16 colunas: a combinação dos bits 1 e 6 da entrada da *S-box* define uma das quatro linhas e a combinação dos bits 2 até 5 define uma das dezasseis colunas. Uma vez que cada *S-box* tem a sua tabela, são necessárias 8 tabelas

para definir a saída dessas *S-boxes*. Essas tabelas estão representadas nas tabelas 3 a 10 (os valores das entradas e das saídas estão como valores decimais).

Tabela 3 *S-box* 1

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Tabela 4 *S-box* 2

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

Tabela 5 *S-box* 3

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

Tabela 6 S-box 4

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

Tabela 7 S-box 5

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

Tabela 8 S-box 6

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	10	00	08	13

Tabela 9 *S-box* 7

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

Tabela 10 S-box 8

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	10	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	10	15	03	05	08
3	02	01	14	07	04	10	08	13	15	12	09	09	03	05	06	11

A última operação da função DES é uma permutação (*D-box* linear – ver figura 6) com uma entrada de 32 bits e uma saída de 32 bits. A permutação é realizada tendo em conta uma tabela predefinida (tabela 11), em que, por exemplo, o vigésimo bit de entrada torna-se o terceiro bit de saída.

Tabela 11 *D-box* linear

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Na figura 7 pode ver-se a estrutura da encriptação e desencriptação do DES mais em pormenor, permitindo identificar melhor as suas semelhantes. Como se pode perceber facilmente, a desencriptação é um processo inverso ao da encriptação. Conclui-se também que, tanto na encriptação como na desencriptação, a última ronda (ronda 16) só tem um misturador e não tem nenhum swapper. As rondas em si não estão alinhadas estando, no entanto, os misturadores e os swappers alinhados. Convém ainda referir que as chaves por ronda (K1 a K16) devem ser aplicadas por ordem inversa na desencriptação.

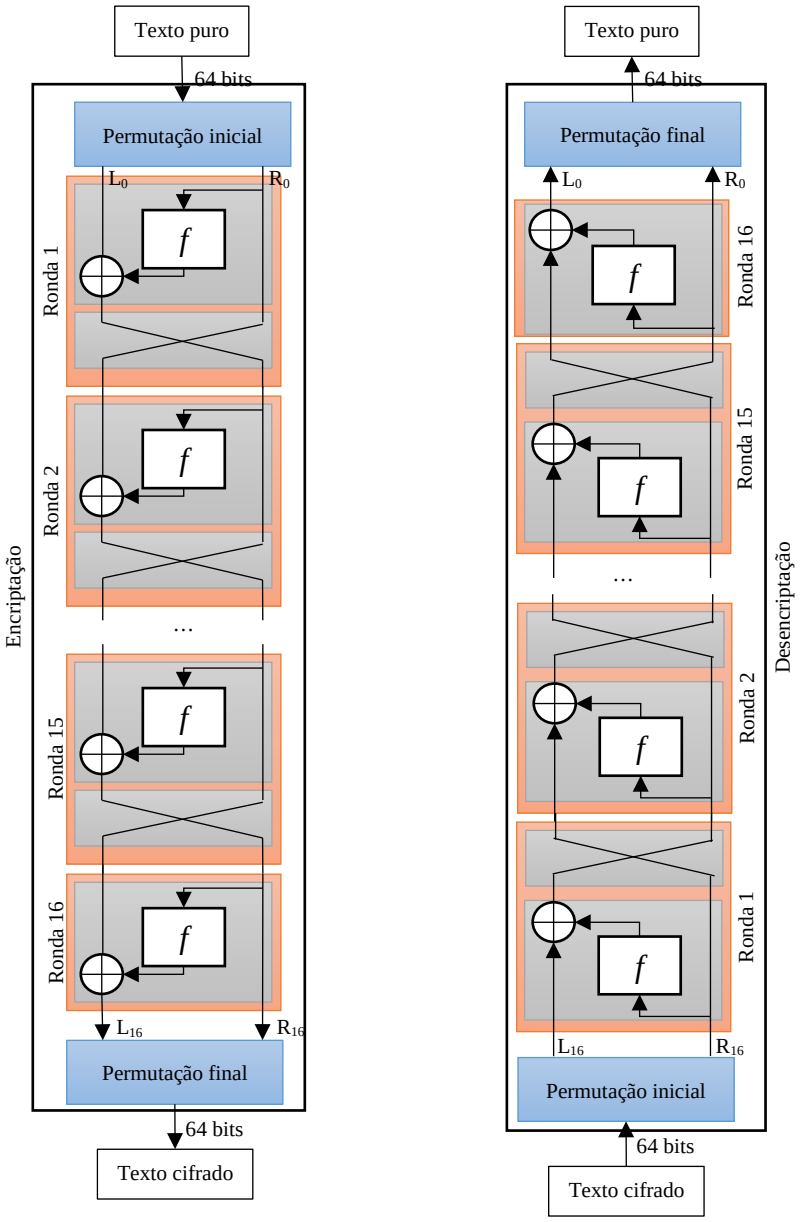


Figura 7 Encriptação e desencriptação do DES

2.2.1. GERADOR DA CHAVE DE RONDA

O gerador da chave de ronda cria as 16 chaves de 48 bits para cada ronda a partir da chave original da cifra. A chave da cifra é constituída por 64 bits, sendo 8 bits de paridade que são eliminados antes do processo de geração da chave propriamente dito. O esquema do gerador da chave de ronda é mostrado na figura 8.

O gerador da chave de ronda começa por eliminar os bits de paridade da chave original (8, 16, 24, 32, …, 64) e permuta os restantes de acordo com a tabela 12. Estes 56 bits restantes são utilizados para gerar as chaves de ronda.

Tabela 12 Tabela de eliminação dos bits de paridade

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Depois da eliminação dos bits de paridade, a chave é dividida em 2 partes de 28 bits e cada parte é deslocada 1 ou 2 bits para a esquerda. Nas rondas 1, 2, 9 e 16 é deslocada 1 bit para a esquerda, nas restantes rondas é deslocada 2 bits. Depois as duas partes são novamente combinadas formando uma parte de 56 bits. A seguir a chave é encurtada para 48 bits pela *D-box* de compressão de acordo com a tabela 13. A chave resultante é a chave utilizada na chave de ronda.

Tabela 13 *D-box* de compressão

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

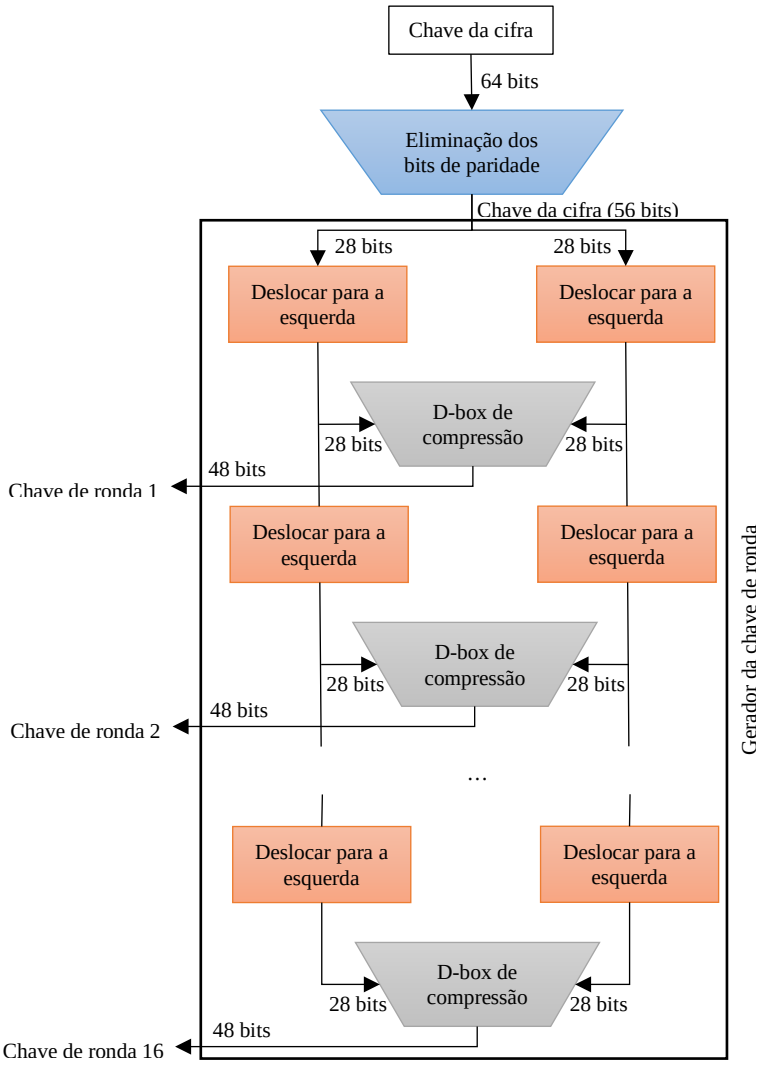


Figura 8 Gerador da chave de ronda

2.2.2. PROBLEMAS DO DES

O DES desde praticamente o seu lançamento tem sido muito estudado devido à sua popularidade e, por isso, ao longo do tempo foram encontrados alguns pontos fracos do algoritmo. Começando pelas *S-boxes* há 3 pontos fracos conhecidos:

1. Na *S-box* 4, os últimos 3 bits de saída podem ser derivados da mesma maneira como o primeiro bit de saída através da complementaridade de alguns bits de entrada;

2. Duas entradas escolhidas especificamente para uma matriz *S-box* podem criar a mesma saída;

3. É possível obter o mesmo resultado de uma ronda alterando os bits em apenas 3 *S-boxes* vizinhas.

O DES também tem um ponto fraco na função DES, mais precisamente na *D-box* de expansão: a cada série de 4 bits o primeiro e o quarto bits são repetidos. Em relação às permutações inicial e final, não está claro pelos criadores o porquê de elas existirem, pois não têm nenhum benefício a nível da segurança do DES.

Também foram identificados vários problemas na chave da cifra. O problema mais grave é o tamanho da chave, que é só de 56 bits. Num ataque de força bruta a um determinado bloco de texto cifrado, é apenas preciso verificar 2^{56} chaves.

É possível ainda identificar outro problema da chave da cifra: em cada 2^{56} chaves da cifra obtém-se 4 chaves fracas. Isto acontece na eliminação dos bits de paridade, antes de gerar a chave de ronda, quando a chave da cifra é constituída por tudo 0s ou 1s ou metade por 0s ou 1s e ainda as chaves da cifra FEFE. Na tabela 14 são mostradas essas chaves da cifra e como ficam após a eliminação dos bits de paridade.

Tabela 14 Chaves fracas

Chaves antes da eliminação dos bits de paridade (64 bits)				**Chave atual (56 bits)**	
0101	0101	0101	0101	0000000	0000000
1F1F	1F1F	0E0E	0E0E	0000000	FFFFFFF
E0E0	E0E0	F1F1	F1F1	FFFFFFF	0000000
FEFE	FEFE	FEFE	FEFE	FFFFFFF	FFFFFFF

As chaves de ronda criadas a partir destas chaves fracas são as mesmas e têm o mesmo padrão que a chave da cifra. A razão é que o algoritmo do gerador das chaves de ronda divide em duas partes a chave da cifra e a deslocação e a permutação de um bloco não alteram o bloco se ele for de tudo 1s ou 0s. As chaves fracas têm como desvantagem a possibilidade de se obter o texto original encriptando novamente o resultado da encriptação (ou seja, encriptar o texto cifrado). Inversamente também é possível obter o mesmo resultado, isto é, ao decifrar duas vezes seguidas obtém-se o bloco original. Isto é possível, porque cada chave fraca é inversa da própria. Para além das chaves fracas também há 6 pares de chaves semi--fracas. Estes pares são mostrados na tabela 15 e são os pares antes da eliminação dos bits de paridade (64 bits).

Tabela 15 Chaves semi-fracas

Primeira chave do par				Segunda chave do par			
01FE	01FE	01FE	01FE	FE01	FE01	FE01	FE01
1FE0	1FE0	0EF1	0EF1	E01F	E01F	F10E	F10E
01E0	01E1	01F1	01F1	E001	E001	F101	F101
1FFE	1FFE	0EFE	0EFE	FE1F	FE1F	FE0E	FE0E
011F	011F	010E	010E	1F01	1F01	0E01	0E01
E0FE	E0FE	F1FE	F1FE	FEE0	FEE0	FEF1	FEF1

Cada chave semi-fraca cria apenas duas chaves de ronda diferentes e cada uma é repetida 8 vezes. Além disso, cada par cria as mesmas chaves de ronda, mas com diferentes ordens. Por exemplo, a chave de ronda 1 do primeiro conjunto é a mesma do que a chave de ronda 16 do segundo conjunto, a chave de ronda 2 do primeiro conjunto é a mesma do que a chave de ronda 15 do segundo conjunto, etc. Isto mostra que as chaves semi-fracas são inversas umas das outras.

Por último, também há 48 chaves fracas possíveis. Uma chave fraca possível é uma chave que cria só 4 chaves de ronda distintas.

2.2.3. CRIPTOANÁLISE

Os criptoanalistas estudam os sistemas de criptografia, com o objetivo de tentar entender como eles funcionam e se existem falhas que lhes permite quebrar a cifra, com ou sem chave. Dito de outra forma, a criptoanálise estuda um conjunto de técnicas, com base em análises matemáticas, para atacar os pontos fracos dos algoritmos criptográficos. O objetivo é derrotar algumas propriedades de segurança que os algoritmos devem cumprir, por exemplo, desencriptar dados que foram encriptados simetricamente com muito menos esforço do que o que seria necessário para testar todas as chaves possíveis. Sabendo deste conceito, será analisada a seguir a criptoanálise contra o DES.

Ataque de força bruta

O ataque de força bruta é um ataque que tenta todas as chaves possíveis até encontrar a chave correta. Na pior das hipóteses, o ataque irá exigir 2^n passos para um tamanho da chave de n bits e, em média, exigirá 2^{n-1} passos. Para o DES, isto significa que exigirá 2^{56} operações de encriptação para o pior caso ou 2^{55} operações de encriptação em média. Sabendo que o ataque tenta todas as chaves até encontrar a correta, ele torna-se interessante no campo da

paralelização. Com N processadores é possível encontrar a chave, aproximadamente, N vezes mais rápido do que com 1 processador. Isto requer algum tipo de entidade de controlo centralizado para distribuir o trabalho e monitorizar o processo de pesquisa.

A partir de Janeiro de 1997, o RSA patrocinou uma série de desafios ao longo de 3 anos para reforçar a ideia das limitações do DES. O primeiro desafio foi resolvido pela equipa DESCHALL, liderada por *Rocke Verser*, *Matt Curtin* e *Justin Dolske*. Esta equipa criou uma malha constituída por voluntários, universidades e corporações em todo o mundo que doaram os seus ciclos de CPU não utilizados para realizar o trabalho. Eles conseguiram encontrar a chave em 90 dias. O segundo desafio foi lançado no dia 19 de Dezembro de 1997 e no dia 26 de Fevereiro de 1998 foi anunciado o vencedor: *distributed.net* com a cooperação da *Electronic Frontier Foundation* (EFF). Este vencedor conseguiu encontrar a chave em 39 dias e contou com um total de 22000 participantes e mais de 50000 CPUs. O terceiro desafio foi lançado no dia 22 de Dezembro de 1998 e no dia 19 de Janeiro de 1999 foi anunciado o vencedor. Novamente o desafio foi vencido pela *distributed.net* em cooperação com a EFF. Eles conseguiram encontrar a chave em apenas 22 horas, o que mostrou nesse ano a enorme fragilidade do DES.

A primeira máquina criada para atacar o DES através do ataque de força bruta foi uma máquina contruída pela EFF em 1998. Esta máquina tinha o poder de quebrar o DES no pior dos casos em cerca de 9 dias ou em média de 4,5 dias e custava 250000 dólares. A próxima máquina criada especificamente para atacar o DES foi a COPACOBANA em 2006 [2]. Esta máquina foi criada por equipas das universidades de *Bochum* e *Kiel* da Alemanha. A COPACOBANA hospedava 120 *Field Programmable Gate Arrays* (FPGAs) de baixo custo e conseguia realizar a pesquisa exaustiva da chave ao DES em menos de 9 dias, em média. Na altura, esta máquina tinha um custo de 10000 dólares. Foi significativa a redução do custo em relação à máquina da EFF, com uma redução de 25 vezes do custo da máquina. Desde 2007, que a empresa *SciEngines GmbH*, uma empresa *spin-off* criada pelos dois parceiros do projeto COPACOBANA, têm melhorado a COPACOBANA com FPGAs mais recentes. Logo em 2008, eles conseguiram reduzir o tempo para quebrar o DES para menos de um dia.

O ataque de força bruta é chamado assim, porque ele não exige muita inteligência no processo de ataque, pois ele simplesmente tenta uma chave após outra. Contudo, existem

ataques criptoanalíticos mais rápidos do que o ataque de força bruta, mas exigem um número irrealista de textos puros conhecidos ou escolhidos para levar a cabo esses ataques, o que se tornam impraticáveis. Estes ataques podem quebrar as 16 rondas do DES e são designados por criptoanálise diferencial, criptoanálise linear e ataque de *Davies*.

Criptoanálise diferencial

A criptoanálise diferencial não era conhecida publicamente até ao ano de 1990. A primeira vez em que esta criptoanálise foi falada foi em 1990 pelo *Murphy* para atacar a cifra de bloco *Fast data Encipherment ALgorithm* (FEAL) [42]. Depois deste artigo, seguiram-se vários artigos publicados por *Biham* e *Shamir*, onde demonstraram esta forma de ataque contra vários algoritmos de criptografia e funções *hash*. Contudo, o ataque que ficou mais conhecido foi o ataque que eles criaram ao DES.

A criptoanálise diferencial foi o primeiro ataque publicado capaz de quebrar o DES com menos complexidade do que o ataque de força bruta (complexidade de 2^{55} chaves) [66]. Segundo o trabalho de *Biham* e *Shamir* é possível quebrar o DES com um esforço na ordem de 2^{47} encriptações, exigindo 2^{47} textos puros escolhidos – o atacante escolhe textos puros aleatórios para serem criptografados e obter os correspondentes textos cifrados, o objetivo é obter informações adicionais que reduzem a segurança do esquema utilizado na cifra e, no pior dos casos, descobrir a chave secreta usada. Apesar de 2^{47} textos puros serem bem menos do que a complexidade 2^{55} exigida pelo ataque de força bruta, a necessidade de o atacante ter que encontrar 2^{47} textos puros torna este ataque apenas teórico e inviável na prática.

Embora a criptoanálise diferencial seja uma ferramenta poderosa de ataque, ela não é assim tão eficaz contra o DES. A possível razão de isto acontecer, de acordo com um membro da equipa da IBM que projetou o DES, é que a criptoanálise diferencial já era conhecida pela equipa desde 1974. Por isso, houve o cuidado de reforçar o DES contra a criptoanálise diferencial, que teve um papel importante na conceção das *S-boxes* e as *P-boxes*. A título de exemplo e como relatado por *Biham* e *Shamir* [5], a criptoanálise diferencial contra um algoritmo LUCIFER de 8 rondas requer apenas 256 textos puros escolhidos, enquanto que um ataque a uma versão de 8 rondas do DES requer 2^{14} textos puros escolhidos.

O ataque de criptoanálise diferencial é complexo. A lógica por detrás do ataque é analisar a evolução dos pares de blocos de texto puro ao longo de cada ronda da cifra. No DES, o bloco

de texto puro é dividido em 2 depois da permutação inicial e antes das rondas: m_0 e m_1. Cada ronda do DES mapeia a entrada do lado direito para a saída do lado esquerdo e define a saída do lado direito para ser uma função de entrada do lado esquerdo e a subchave para essa ronda. Assim, em cada ronda, apenas um bloco de 32 bits é criado. Se for dado um nome diferente a cada bloco de m_1, então as metades das mensagens intermédias estão relacionadas através da equação 1:

$$m_{i+1} = m_{i-1} \oplus f(K_i), i = 1, 2, \dots, 16 \quad (1)$$

Na criptoanálise diferencial começa-se com duas mensagens (textos puros) m e m', com uma diferença XOR conhecida $\Delta m = m \oplus m_i = m_i \oplus \Delta m_{i+1}$, onde Δm_{i+1} se calcula através da equação 2:

$$\begin{aligned} \Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} = \quad (2) \\ &= [m_{i-1} \oplus f(m_i, K_i)] \oplus [m'_{i-1} \oplus f(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [f(m_i, K_i) \oplus f(m'_i, K_i)] \end{aligned}$$

Sabendo destes dados, o criptoanalista supõe que muitos pares de entrada para f com a mesma diferença produzirá a mesma diferença de saída se a mesma subchave for usada. Explicando de outra maneira: diga-se que X pode causar Y com uma probabilidade p, se para uma parte de p dos pares a entrada do XOR é X, a saída do XOR é igual a Y. Pode-se supor que há uma série de valores de X que têm uma elevada probabilidade de causar uma diferença de saída específica. Logo, se for conhecido Δm_{i-1} e Δm_i com alta probabilidade, então conhece-se com alta probabilidade Δm_{i+1}. Consequentemente, se o número de tais diferenças for determinado é possível determinar a subchave usada na função f.

O principio da criptoanálise diferencial é usar estas considerações para uma única ronda. O procedimento é começar com 2 mensagens de texto puro m e m' com uma determinada diferença e rastrear através de um padrão de diferenças após cada ronda para produzir uma provável diferença para o texto cifrado. Na verdade, existem dois padrões prováveis de diferenças para as duas metades de 32 bits: ($\Delta m_{17} \,//\, m_{16}$). Sabendo disto, submete-se m e m' para a encriptação para determinar a diferença atual da chave desconhecida e comparar o resultado com a diferença provável (equação 3). Se houver uma correspondência, supõe-se todos os padrões prováveis em todas as rondas intermédias estão corretos. Com este

pressuposto é possível fazer algumas deduções sobre os bits da chave. Este procedimento tem de ser repetido muitas vezes para determinar todos os bits da chave.

$$E(K, m) \oplus E(m') = (\Delta\, m_{17} \,||\, m_{16}) \quad (3)$$

Criptoanálise linear

Em 1992, *Matsui* e *Yamagishi* introduziram a ideia da criptoanálise linear num ataque contra a cifra de bloco FEAL [6]. As técnicas utilizadas neste ataque foram aperfeiçoadas pelo *Matsui* em 1993 e usadas contra o DES, em um ataque teórico contra as 16 rondas, exigindo 2^{47} pares de texto puro conhecido / texto cifrado [9]. Depois, em 1994, com mais algum trabalho de aperfeiçoamento do ataque, *Matsui* conseguiu atacar as 16 rondas do DES usando 2^{43} pares de texto puro conhecido / texto cifrado [11].

A caraterística mais interessante da criptoanálise linear é que ela é um ataque de texto puro conhecido, enquanto a criptoanálise diferencial é um ataque de texto puro escolhido e, como tal, é uma ameaça mais prática do que a criptoanálise diferencial. Contudo, um ataque com a criptoanálise linear contra as 16 rondas do DES ainda requer uma grande quantidade de texto puro conhecido.

A ideia base por trás da criptoanálise linear é encontrar alguma aproximação linear que descreva a cifra de blocos iterada. A expressão contém alguns bits do texto puro $\left(P_{i_1} \ldots \mathrm{P_{i_a}}\right)$, do texto cifrado $\left(C_{j_1} \ldots C_{j_b}\right)$ e da chave $\left(K_{k_1} \ldots K_{k_u}\right)$. Essa expressão está descrita na equação 4, onde foi substituído, para simplificar, os XORs dos bits do texto puro $\left(P_{i_1} \oplus \ldots \oplus P_{i_a}\right)$ por $P_{[\chi P]}$ e também foram substituídos da mesma forma os bits do texto cifrado e da chave:

$$P_{[\chi P]} \oplus C_{[\chi C]} = K_{[\chi K]} \quad (4)$$

Se a equação 4 for correta com probabilidade $p = \frac{1}{2} + \epsilon$ para textos puros escolhidos aleatoriamente e uma chave fixa, então diz-se que ela tem tendência para ϵ. O criptoanalista ao recolher pares de texto puro conhecido / texto cifrado pode fazer uma estimativa para o valor do $K_{[\chi K]}$. Desde que $\epsilon \neq 0$, a estimativa torna-se cada vez mais confiável à medida que o criptoanalista recolha mais pares de texto puro conhecido / texto cifrado.

Deve-se salientar que o valor esperado do lado esquerdo é $\frac{1}{2} + \epsilon$ se $K_{[\chi K]} = 1$ e $\frac{1}{2} - \epsilon$ se $K_{[\chi K]} = 0$. Sob determinados pressupostos heurísticos, o criptoanalista está a tentar

distinguir uma distribuição com média $\frac{1}{2} + \epsilon$ e uma variação $\frac{1}{4} - \epsilon^2$ para $K_{[\chi K]} = 1$ ou com média $\frac{1}{2} - \epsilon$ e uma variação $\frac{1}{4} - \epsilon^2$ para $K_{[\chi K]} = 0$. O criptoanalista tem de ter um número suficiente de pares de texto puro / texto cifrado para se certificar que a distinção está sendo feita corretamente. Assim, quanto menor for o valor de ϵ mais pares de texto puro / texto cifrado são necessários para dar o mesmo nível de confiança de que a identificação está correta.

Ataque de *Davies* melhorado

O ataque de *Davies* melhorado foi criado por *Eli Biham* e *Alex Biryukov* em 1994 [14]. Eles conseguiram melhorar o ataque de *Davies*, de modo a conseguirem quebrar o DES completo (16 rondas) mais rápido do que o ataque de força bruta. Este ataque requer 2^{50} de dados e 2^{50} passos para análise. Este ataque é o terceiro ataque mais bem-sucedido contra o DES, depois da criptoanálise diferencial e linear.

Para se perceber melhor o ataque de *Davies* melhorado convém introduzir primeiro o ataque de *Davies* original. O ataque de *Davies* foi criado em 1987 pelo *Donald Davies* e é um ataque específico contra o DES, que se baseia na não uniformidade da distribuição das saídas dos pares de *S-boxes* adjacentes. Teoricamente, com este ataque é possível ganhar até 16 bits de paridade da chave com este método. Contudo, o ataque de *Davies* não é prático de se aplicar, pois a distribuição resultante das saídas dos pares de *S-boxes* adjacentes do DES é muito uniforme. A variante baseada no melhor par (S_7 e S_8) exige $2^{56,6}$ de textos puros conhecidos e encontra 2 bits de paridade da chave com uma taxa de sucesso 95,5%.

A operação de expansão do DES duplica os bits de dados para entrar em duas *S-boxes* adjacentes. Então, cada par de *S-boxes* adjacentes partilha dois bits de dados. Estes bits, antes de entrarem nas *S-boxes*, é realizada uma operação XOR com diferentes bits da chave. Como resultado, a saída de um dos pares adjacentes de *S-boxes* têm distribuição não uniforme. *Davies* descobriu que essa distribuição depende apenas da paridade de quatro bits da chave que são misturados com os bits de dados partilhados. Se esta paridade for denominada por p_1, o valor médio dos vários valores de distribuição por $E(D_1)$ e as saídas do par *S-box* por x (para a *S-box* esquerda) e y (para a *S-box* direita), então a distribuição de saída de um par de *S-boxes* pode ser escrita através da equação 5:

$$D_1(x, y, p_1) = E(D_1) + (-1)^{p_1}.d_1(x, y) \quad (5)$$

O XOR das saídas das funções f dos 8 pares de rondas podem ser calculadas pelo XOR da metade direita do texto puro com a metade esquerda do texto cifrado e aplicando a permutação inversa P^{-1}. *Davies* chegou à conclusão que as distribuições XOR das saídas dos pares de *S-boxes* têm uma equação semelhante à equação 6, onde p_n é a paridade dos 4n bits da subchave que são misturados com os bits dos dados das n rondas pares e $E(D_n) = 2^{10n-8}$ é a média de distribuição:

$$D_n(x, y, p_n) = E(D_n) + (-1)^{p_n}.d_n(x, y) \quad (6)$$

$D_n(x,y,p_n)$ pode ser calculado recorrendo às equações 7 e 8:

$$D_n(x, y, 0) = \sum_{\substack{x_1 \oplus x_2 = x \\ y_1 \oplus y_2 = y}} D_{n-1}(x_1, y_1, 0).D_1(x_2, y_2, 0); \quad \text{n = 2 … 8} \quad (7)$$

$$D_n(x, y, 1) = 2E(D_n) - D_n(x, y, 0) \quad (8)$$

A tabela 16 mostra as complexidades do ataque de *Davies* em diferentes pares de *S-boxes* e em diferentes rondas. Estas complexidades descritas na tabela 16 são para encontrar 2 bits de paridade para as rondas pares e 1 bit de paridade para as rondas ímpares. O melhor par de *S-boxes* (S7/8) requer $2^{56,6}$ de textos puros conhecidos para encontrar 2 bits de paridade. Devido à necessidade de o atacante precisar de muitos textos puros conhecidos para realizar o ataque, este ataque torna-se impraticável.

Tabela 16 As complexidades do ataque de *Davies*

Rondas	**Distribui.**	**S1/2**	**S2/3**	**S3/4**	**S4/5**	**S5/6**	**S6/7**	**S7/8**	**S8/1**
2,3	D_1	$2^{6,4}$	$2^{6,1}$	$2^{8,8}$	$2^{6,7}$	$2^{7,4}$	$2^{7,1}$	$2^{6,2}$	$2^{7,7}$
4,5	D_2	$2^{16,3}$	$2^{15,7}$	$2^{20,4}$	$2^{16,7}$	$2^{17,6}$	$2^{16,8}$	$2^{14,5}$	$2^{18,5}$
6,7	D_3	$2^{25,2}$	$2^{24,9}$	$2^{31,4}$	$2^{26,0}$	$2^{27,0}$	$2^{25,4}$	$2^{21,8}$	$2^{28,6}$
8,9	D_4	$2^{33,6}$	$2^{33,9}$	$2^{42,3}$	$2^{35,1}$	$2^{36,1}$	$2^{33,7}$	$2^{28,9}$	$2^{38,5}$
10,11	D_5	$2^{41,8}$	$2^{42,8}$	$2^{53,1}$	$2^{44,1}$	$2^{45,0}$	$2^{41,8}$	$2^{35,9}$	$2^{48,2}$
12,13	D_6	$2^{49,9}$	$2^{51,6}$	$2^{64,0}$	$2^{52,9}$	$2^{53,9}$	$2^{49,9}$	$2^{42,8}$	$2^{57,9}$
14,15	D_7	$2^{57,9}$	$2^{60,5}$	$2^{74,8}$	$2^{61,8}$	$2^{62,8}$	$2^{57,9}$	$2^{49,7}$	$2^{67,6}$
16	D_8	$2^{66,0}$	$2^{69,3}$	$2^{85,6}$	$2^{70,6}$	$2^{71,6}$	$2^{66,0}$	$2^{56,6}$	$2^{77,3}$

Eli Biham e o *Alex Biryukov* melhoraram este ataque em 1994 [14]. Eles descobriram que a distribuição D_7 pode ser usada em vez da D_8. A distribuição D_7 é menos uniforme do que a

D_8 e, por isso, é necessário um menor número de textos conhecidos. Para se usar a D_7 tem que se descartar uma ronda do DES para ser possível adivinhar todos os possíveis valores dos bits da chave do par de *S-boxes* e calcular a possível distribuição que resulta para cada valor dos bits da chave que entra no par de *S-boxes* na última ronda depois do XOR dos bits do texto puro e encriptado com a saída das *S-boxes*. Eles estudaram apenas o par de *S-boxes* S7/8 por ser menos uniforme, como já foi visto antes na tabela 16. Todos os outros pares resultam numa complexidade maior do que o ataque de força bruta.

O ataque de *Davies* melhorado acrescenta uma ronda ao ataque de *Davies* e pode encontrar 24 bits da chave nas 16 rondas do DES. Este ataque aplica a análise 2 vezes: uma vez às rondas pares e outra vez às rondas ímpares, com a única diferença que a primeira ronda de encriptação é executada supondo que a subchave K_1 tem 2^{12} bits. O ataque calcula a saída do par de *S-boxes* na última ronda, executando uma ronda de descodificação parcial no par de *S-boxes*. O valor de 12 bits da chave que entra nessas *S-boxes* é desconhecida, então é preciso experimentar todas as 2^{12} possibilidades. Como para cada texto cifrado cerca de 1/64 de desencriptação é realizada, a complexidade de ataque é mais do que $2^{49,74}$ x (2^{12}/64) ≈ 2^{56}. Utilizando este método é possível encontrar os 24 bits da chave com uma probabilidade de 0,53. Os outros 32 bits podem ser encontrados pelo ataque de força bruta.

O algoritmo desenhado para realizar o ataque de *Davies* melhorado tem 2 fases: recolha de dados e análise de dados. O atacante para realizar o ataque precisa de 10 bits de texto cifrado. A fase de recolha de dados conta o número de ocorrências de cada valor possível dos 8 bits de distribuição (resultado do XOR do texto puro com o texto cifrado) em conjunto com os 10 bits de texto cifrado (que entram no par de *S-boxes* na última ronda) e sai uma matriz com 2^{18} contadores. A fase de análise de dados começa por calcular as 2^{12} distribuições: para cada valor possível de 12 bits de chave e 10 bits de texto cifrado (α) a entrar no par de *S-boxes*, a saída do par é calculada. Com este resultado de 8 bits é feito um XOR para cada valor possível de 8 bits (β) e a correspondente entrada ($f_K(\alpha) \oplus \beta$). A distribuição que é gerada com estes cálculos é inserida num contador com o valor da chave correspondente à distribuição calculada (α, β). Depois disto, fica-se com 2^{12} distribuições geradas que serão analisadas para encontrar o valor correto da chave. Daqui resulta os 12 bits da subchave da última ronda (K_{16}) mais um bit de paridade da chave. A fase de análise tem uma complexidade de cerca $2^{12} \times 2^{10} \times \frac{1}{64} = 2^{16}$ encriptações DES, mais 2^{30} incrementos de

contador. Esta fase parece complicada, mas corre em poucos minutos numa estação SPARC (*Scalable Processor ARChitecture*).

2.3. *TRIPLE DATA ENCRYPTION STANDARD* (TDES)

Depois de 1990, o avanço do poder computacional permitia quebrar o DES, através do ataque de força bruta, em poucos dias ou até mesmo em horas. Isto fez com que muitos utilizadores do DES começassem a temer a segurança do algoritmo. Contudo, os utilizadores não queriam mudar de algoritmo, porque para mudar de algoritmo é necessário muito tempo e dinheiro para serem incorporados nos seus sistemas. Então, chegou-se à conclusão que o objetivo a seguir não era abandonar o DES, mas mudar a maneira como o DES era usado. Isto levou à criação de duas variantes do TDES: *3-key Triple DES* (3TDES) e *2-key Triple DES* (2TDES).

Para se usar o 3TDES, primeiro tem que se gerar uma chave K de 3TDES, que consiste em três chaves DES diferentes: K_1, K_2 e K_3. Isto significa que a chave real 3TDES tem um tamanho de 3 x 56 = 168 bits. Depois de se ter as 3 chaves, pode-se realizar o processo de encriptação:

1. Encriptar os blocos de texto puro, usando o DES normal com a chave K_1;
2. Desencriptar a saída do passo 1, usando o DES normal com a chave K_2;
3. Encriptar a saída do passo 2, usando o DES normal com a chave K_3.

Para desencriptar, o processo é o inverso da encriptação:

1. Desencriptar os blocos de texto cifrado, usando o DES normal com a chave K_3;
2. Encriptar a saída do passo 1, usando o DES normal com a chave K_2;
3. Desencriptar a saída do passo 2, usando o DES normal com a chave K_1.

Estes processos de encriptação e desencriptação permitem, se os utilizadores quiserem, que o 3TDES tenha compatibilidade com o DES normal, usando uma aplicação 3TDES para definir K_1, K_2 e K_3 com o mesmo valor.

O 2TDES é idêntico ao 3TDES, só que o 2TDES apenas trabalha com 2 chaves (K_1 e K_2), ou seja, nos processos de encriptação e desencriptação do 2TDES o K_3 é substituído pelo K_1. Como o 2TDES trabalha só com 2 chaves, o tamanho real da chave é 2 x 56 = 112 bits. Então, o processo de encriptação do 2TDES é o seguinte:

1. Encriptar os blocos de texto puro, usando o DES normal com a chave K_1;
2. Desencriptar a saída do passo 1, usando o DES normal com a chave K_2;
3. Encriptar a saída do passado 2, usando o DES normal com a chave K_1.

Os sistemas 3TDES são muito mais seguros do que o DES normal, porém o 3TDES é substancialmente mais lento do que o DES normal.

2.3.1. CRIPTOANÁLISE

Primeiro convém referir que o TDES com 3 chaves independentes não é tão seguro como um sistema criptográfico com uma chave 3 vezes maior, ao contrário do que muita gente possa pensar. O TDES só com 2 chaves é significativamente mais fraco do que o TDES com 3 chaves. Este algoritmo TDES é resistente contra a criptoanálise linear e diferencial e contra o ataque de força bruta. Contudo, o ganho de segurança obtido por fazer várias encriptações não é tão elevado como se poderia esperar, principalmente por causa dos ataques *meet-in-the-middle*. No entanto, estes ataques não representam uma ameaça imediata, porque atualmente necessitam de uma quantidade absurda de recursos.

Ataque *meet-in-the-middle*

O tipo de ataque mais conhecido contra este tipo de cifras é o ataque *meet-in-the-middle*. Este ataque foi criado por *Diffie* e *Hellman* em 1977 [67], utiliza uma relação de compromisso de espaço-tempo, para reduzir em muito a complexidade de quebra das cifras que têm um esquema múltiplo de encriptação. Por outras palavras, é uma técnica que tenta encontrar valores para possíveis chaves que encriptam um texto puro até uma parte da encriptação e também possíveis chaves parciais que desencriptam o texto cifrado. Depois são guardados numa tabela esses possíveis valores das chaves e quando houver dois valores iguais tem-se uma chave candidata.

Para se perceber como funciona este ataque vai ser explicado a seguir com um exemplo para a versão de 3 chaves do TDES. Se E_K e D_K representa as funções de encriptação e desencriptação do DES com uma chave $K \in \{0,1\}^{56}$ e se C e P representam o texto cifrado e o texto puro respetivamente, então o TDES com as 3 chaves pode ser escrito usando as equações 9 e 10:

$$C = E_{K_3}\left(D_{K_2}\left(E_{K_1}(P)\right)\right) \quad (9)$$

$$P = D_{K_1}\left(E_{K_2}\left(D_{K_3}(C)\right)\right) \quad (10)$$

A partir da equação 9, pode-se desencriptar o texto cifrado usando a desencriptação da terceira chave:

$$D_{K_3}(C) = D_{K_3}\left(E_{K_3}\left(D_{K_2}\left(E_{K_1}(P)\right)\right)\right) \Leftrightarrow$$

$$\Leftrightarrow D_{K_3}(C) = D_{K_2}\left(E_{K_1}(P)\right)$$

Com estes dados, pode-se descrever o ataque *meet-in-the-middle* a este tipo de cifras usando as equações 11 e 12:

$$H = \left\{\left(K, D_K(P)\right) : K \in \{0,1\}^{56}\right\} \quad (11)$$

$$S = \left\{(K_a, K_b, K_c) : K_a, K_b, K_c \in \{0,1\}^{56} \wedge \left(K_c, D_{K_b}\left(E_{K_a}(C)\right)\right) \in H\right\} \quad (12)$$

Então, para realizar o ataque *meet-in-the-middle* é necessário primeiro calcular todas as encriptações possíveis do texto puro P usando D_K e armazenar numa tabela de pesquisa H. Depois, calcular todas as desencriptações possíveis do texto cifrado C usando E_{Ka} e D_{Kb} e comparar com a tabela de pesquisa H a ver se há alguma associação. As interseções entre os dois conjuntos irão dar as três chaves corretas. Se resultar vários conjuntos de três chaves nessas interseções, então pode-se testar com pares de texto puro – texto cifrado e descobrir rapidamente quais são as três chaves corretas.

Para construir a tabela de pesquisa serão necessárias 2^{56} passos e um espaço de memória de 2^{56}. Seguidamente serão necessários 2^{112} passos para encontrar as chaves candidatas e depois falta isolar a chave correta. Assim, o ataque *meet-in-the-middle* permite achar as chaves K_1,

K_2 e K_3 em cerca 2^{112} encriptações. É por isso, que a criptografia 3TDES é considerada como tendo no máximo 112 bits de segurança efetivos, apesar de ter 168 bits de chave.

O ataque *meet-in-the-middle* contra as cifras 2TDES requer um custo computacional de 2^{57} passos e 2^{56} de memória. Na verdade, o primeiro passo (a construção da tabela) requer 2^{56} passos para construir a tabela de 2^{56} entradas e o segundo passo (encontrar a chave correta) requer no máximo 2^{56}. Assim, tem-se $2^{56} + 2^{56} = 2^{57}$ passos no total.

2.4. *ADVANCED ENCRYPTION STANDARD* (AES)

Em 1977, o *National Institute of Standards and Technology* (NIST) começou um processo para encontrar um substituto para o DES. Nesta altura, o DES já era reconhecido como não sendo seguro, devido aos avanços do poder computacional. O objetivo do NIST era arranjar um substituto para o DES, que poderia ser usado em aplicações de segurança da informação não militares por agências governamentais dos EUA. Contudo, o NIST reconheceu que os utilizadores não governamentais também iriam beneficiar desta futura cifra e que ela seria adotada como um padrão comercial.

Em Janeiro de 1977, o NIST anunciou um concurso aberto e público para o sucessor do DES. Os critérios do concurso não era só a força da criptografia, ou seja, resistência à criptoanálise diferencial e linear, mas também tinha que ser de fácil implementação em *software* e *hardware* e tinha que ter um bom desempenho. O NIST convidou especialistas em criptografia e segurança de dados de todo o mundo para participar no processo de seleção. Foram escolhidos 5 algoritmos de encriptação para serem alvos no processo de seleção. No fim deste processo foi escolhido o algoritmo de encriptação proposto por dois belgas: *Joan Daeman* e *Vincent Rijmen*. Antes da seleção, o algoritmo deles tinha o nome de *Rijndael*, derivado dos seus nomes. Depois da aprovação do algoritmo, o NIST deu o nome de *Advanced Encryption Standard* (AES) e que é o nome usado por este algoritmo até aos dias de hoje. Em 26 de Novembro de 2001, o NIST aprovou formalmente o algoritmo de encriptação AES e publicou como padrão com a designação FIPS-197 [15].

O AES é o algoritmo de encriptação de chave simétrica mais conhecido e utilizado atualmente. Este algoritmo consegue ser, pelo menos, 6 vezes mais rápido do que o TDES. O AES tem um tamanho do bloco de 128 bits, enquanto que as chaves de encriptação do

algoritmo pode ser 128, 192 ou 256 bits. É neste aspeto que o AES se diferencia do *Rijndael*, porque este suporta um tamanho do bloco e da chave que deve ser um múltiplo de 32 bits, desde que tenha um mínimo de 128 bits e um máximo 256 bits. O AES tem todas as especificações e detalhes do d*esign* divulgados e o algoritmo é implementável em C e Java.

2.4.1. FUNCIONAMENTO DO AES

O AES é uma cifra de bloco iterada, ou seja, transforma blocos de tamanho fixo de texto puro em blocos de tamanho idêntico de texto cifrado, através da aplicação repetida de uma transformação invertível, conhecida como função de ronda, e com cada iteração referida como ronda. O AES é baseado na rede de substituição-permutação, em que tem como entradas um bloco de texto puro e uma chave e aplica várias rondas alternadas, constituído por um estágio de substituição seguido por um estágio de permutação, para produzir um bloco de saída de texto cifrado.

Curiosamente, o AES realiza todos os seus cálculos em bytes em vez de bits. Por isso, o AES trata os 128 bits de um bloco de texto puro como 16 bytes. Esses 16 bytes são inseridos em quatro colunas e quatro linhas para serem processados como uma matriz. Diferentemente do DES, o número de rondas do AES é variável e depende do tamanho da chave: 10 rondas para as chaves de 128 bits, 12 rondas para as chaves 192 bits e 14 rondas para as chaves 256 bits. Em cada uma destas rondas é utilizada uma chave de ronda diferente de 128 bits, que é calculada a partir da chave original. Atualmente, o tamanho da chave mais utilizada é a chave de 128 bits. A descrição do algoritmo que será apresentada a seguir descreve esta implementação em concreto.

A estrutura geral do AES pode ser vista na figura 9. A entrada é um bloco único de 128 bits tanto para a encriptação como para a desencriptação e é conhecido como matriz. O bloco é copiado para uma matriz de estado, que é modificada a cada etapa do algoritmo e, no fim, copiada para uma matriz de saída. A chave é expandida para uma matriz com palavras programadas (a matriz w) para formar as chaves de ronda.

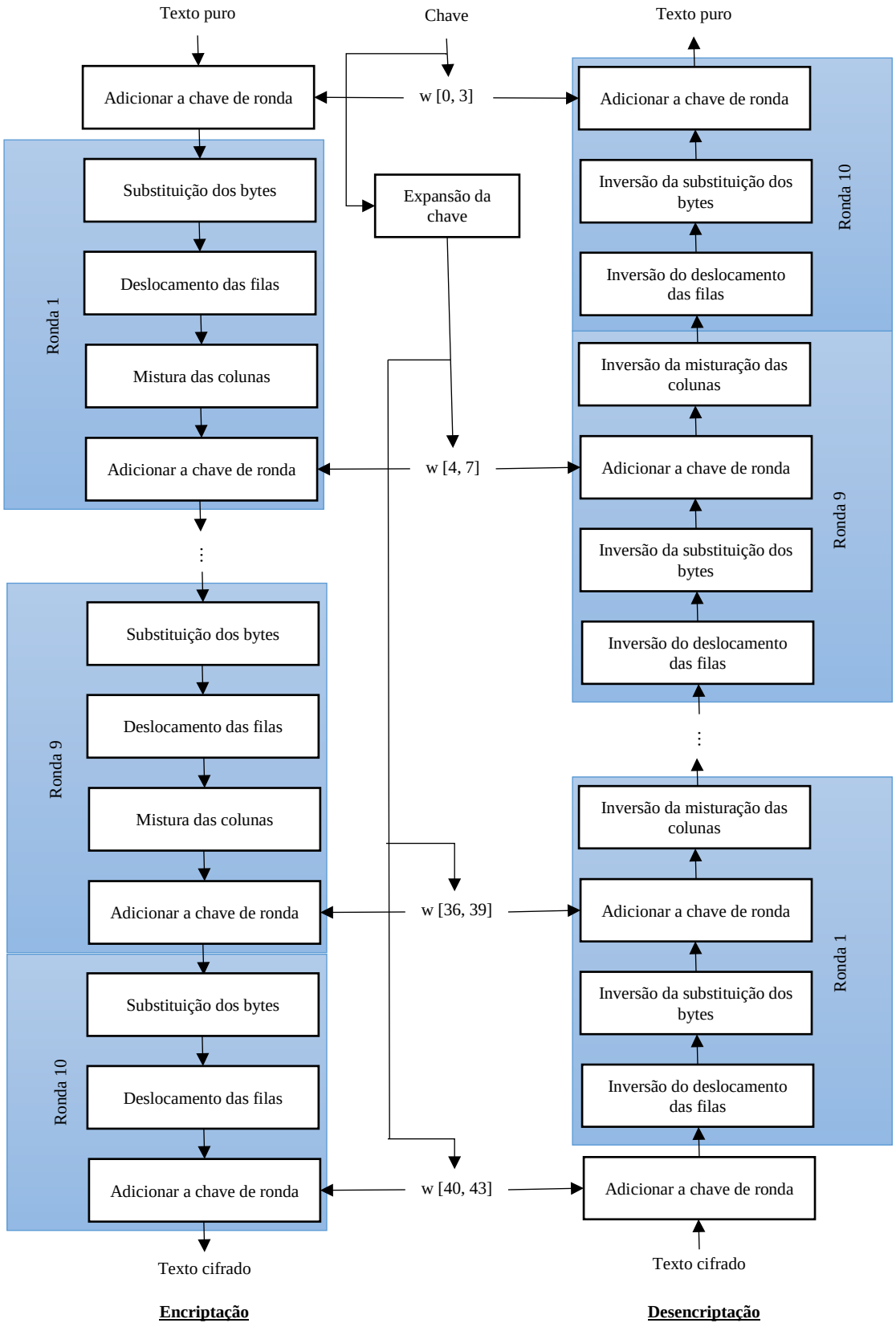


Figura 9 Estrutura geral do algoritmo AES

Funcionamento interno de uma ronda

O algoritmo começa por adicionar a chave de ronda e depois corre 9 rondas com 4 fases e uma décima ronda com 3 fases. Isto aplica-se também para a desencriptação, com a ressalva que em cada fase de uma ronda do algoritmo de desencriptação é o inverso da encriptação. As 4 fases das 9 rondas são: substituição dos bytes, deslocamento das filas, mistura das colunas e adicionar a chave de ronda. Na décima ronda simplesmente deixa de existir a fase da mistura das colunas. A seguir será analisado ao pormenor cada uma destas fases e o algoritmo de expansão da chave.

Expansão da chave

O algoritmo da expansão da chave recebe como entrada a chave original para calcular as 10 subchaves. Elas são guardadas numa matriz de 44 palavras para depois serem usadas em cada ronda na transformação de adicionar a chave de ronda. A figura 10 ilustra como são calculadas as subchaves, em que a palavra mais à esquerda de cada chave de ronda (*W[4i]*) é calculada através da equação 13:

$$W[4i] = W[4(i-1) + g(W[4i-1])] \qquad (13)$$

Onde $i = 1, ..., 10$ e $g()$ é uma função não linear com quatro bytes de entrada e de saída. Para calcular as restantes três palavras de cada subchave usa-se a equação 14, onde $j = 1, 2, 3$:

$$W[4i + j] = W[4i + j - 1] + W[4(i-1) + j] \qquad (14)$$

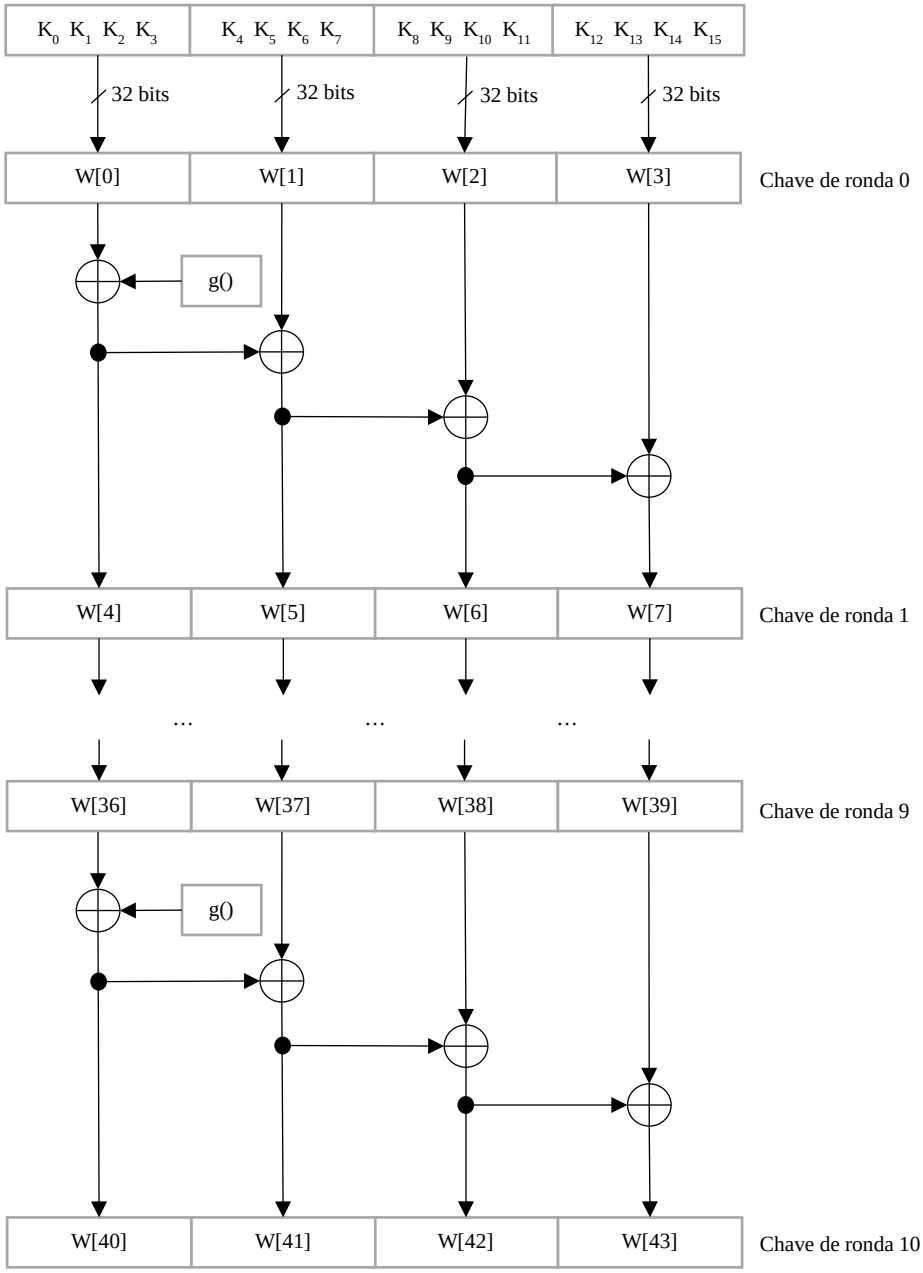


Figura 10 Algoritmo de expansão da chave

A figura 11 mostra uma representação da função *g()*. Ela faz uma rotação de um byte para a esquerda da palavra de entrada, executa uma substituição byte a byte (substituição dos bytes) e adiciona uma constante *round coefficient* (RC) ao byte mais à esquerda da função

$g()$. O RC é um elemento do campo de *Galois*[1] [ou *Galois Field* (GF)] (2^8), ou seja, é um valor de 8 bits. O RC varia de ronda para ronda de acordo com as equações 15, 16, 17 e 18. O objetivo da função $g()$ é adicionar não linearidade e remover as simetrias das chaves.

$$RC[1] = \chi^0 = (00000001)_2 \quad (15)$$

$$RC[2] = \chi^1 = (00000010)_2 \quad (16)$$

$$RC[3] = \chi^2 = (00000100)_2 \quad (17)$$

…

$$RC[10] = \chi^9 = (00110110)_2 \quad (18)$$

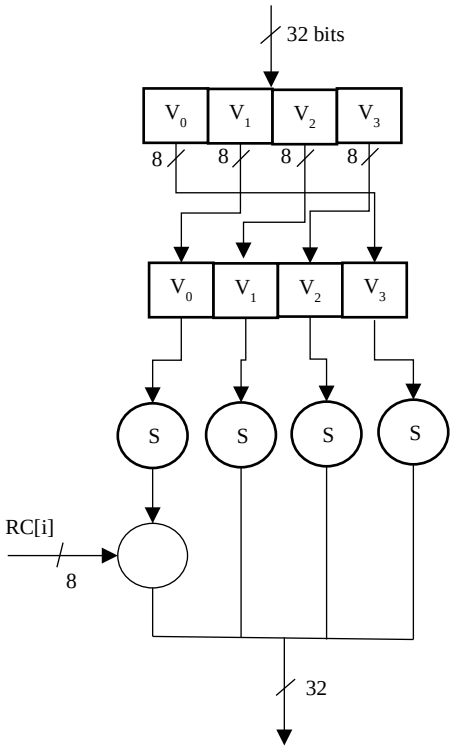


Figura 11 Função $g()$ para uma determinada ronda *i*

[1] O campo de *Galois* ou campo finito é um campo que contém um número finito de elementos e que contém um conjunto de operações de multiplicação, adição, subtração e divisão (excluindo a divisão por zero), satisfazendo as regras da aritmética conhecidas como axiomas de campo.

Substituição dos bytes

A fase da substituição dos bytes consiste numa transformação não linear (representada na figura 12), em que cada byte da matriz de estado é substituído por outro através de uma tabela de referência, que é conhecida por tabela de substituição ou *S-box*. A *S-box* está representada na tabela 17, ela é invertível e é construída através de duas operações: a multiplicativa inversa em GF(2^8) e a transformação *Affine* em GF(2), esta última é dada pela equação 19 para $0 \leq i < 8$:

$$b'_i = b_i \oplus b_{(i+4) mod 8} \oplus b_{(i+5) mod 8} \oplus b_{(i+6) mod 8} \oplus b_{(i+7) mod 8} \oplus c_i \qquad (19)$$

Onde b_i é o i[th] byte de entrada e o c_i é o i[th] byte c, que é uma constante com o valor em hexadecimal 63. Esta transformação também pode ser representada no formato de matriz:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad (20)$$

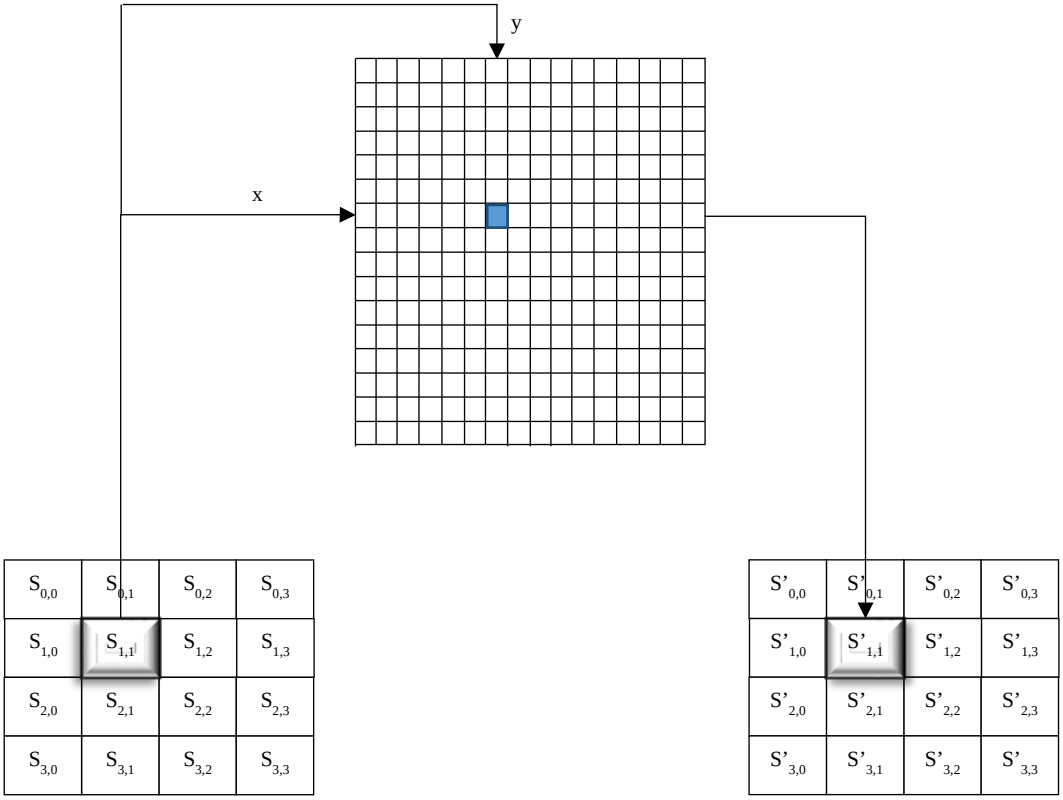


Figura 12 Transformação da fase de substituição dos bytes

A tabela S-box, representada na tabela 17, é uma tabela 16x16 com um byte de entrada e um byte de saída. Quando se quer fazer uma substituição, os quatro bits mais significativos do byte a ser substituído indicam a posição da linha na tabela S-box e os quatro menos significativos a posição da coluna. Estes valores servem de índices para selecionar um byte da tabela. Por exemplo, se houver como byte de entrada o valor hexadecimal 25, este refere-se à linha 2 e coluna 5 da tabela S-box, resultando na saída o valor hexadecimal 3F.

Tabela 17 Tabela da *S-box*

		y														
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
x	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Deslocamento das filas

A transformação do deslocamento das filas consiste numa permutação simples, em que a primeira linha da matriz se mantém inalterada, a segunda linha é deslocada um byte para a esquerda, a terceira linha é deslocada dois bytes para a esquerda e a quarta linha é deslocada três bytes à esquerda. Esta transformação tem como objetivo aumentar a característica de difusão. Na figura 13 está ilustrado o funcionamento desta transformação.

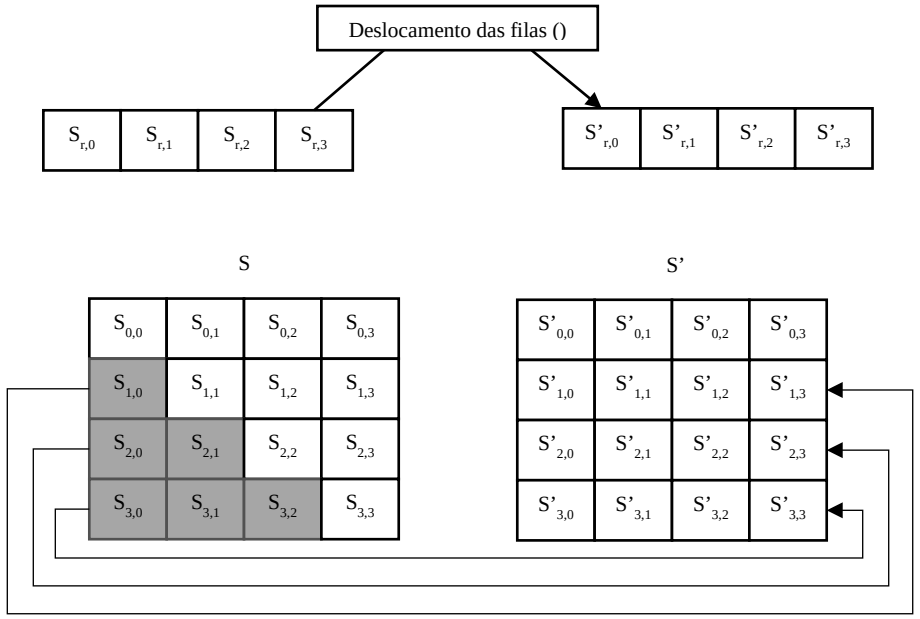


Figura 13 Transformação *ShiftRows*

Mistura das colunas

A transformação da mistura das colunas consiste numa substituição que utiliza aritmética sobre GF(2^8) e trabalha em cada coluna da matriz de estado individualmente. Cada coluna de 4 bytes é trabalhada como um vetor e é multiplicada por uma matriz 4x4. Esta matriz contém valores constantes. Cada elemento da matriz de saída é o resultado da soma dos produtos dos elementos de uma linha com uma coluna. A transformação de uma única coluna da matriz de estado é expressa pelas equações 21, 22, 23 e 24. Ao analisar estas equações, pode-se verificar que cada byte de entrada influencia quatro bytes de saída:

$$S'_{0,j} = (2 \times S_{0,j}) \oplus (3 \times S_{1,j}) \oplus S_{2,j} \oplus S_{3,j} . \quad (21)$$

$$S'_{1,j} = S_{0,j} \oplus (2 \times S_{1,j}) \oplus (3 \times S_{2,j}) \oplus S_{3,j} . \quad (22)$$

$$S'_{2,j} = S_{0,j} \oplus S_{1,j} \oplus (2 \times S_{2,j}) \oplus (3 \times S_{3,j}) . \quad (23)$$

$$S'_{3,j} = (3 \times S_{0,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus (2 \times S_{3,j}) . \quad (24)$$

A combinação do deslocamento das filas com a mistura das colunas faz com que ao fim de apenas três rondas cada byte da matriz dependa de todos os 16 bytes do texto puro. A figura 14 mostra como funciona a transformação da mistura das colunas.

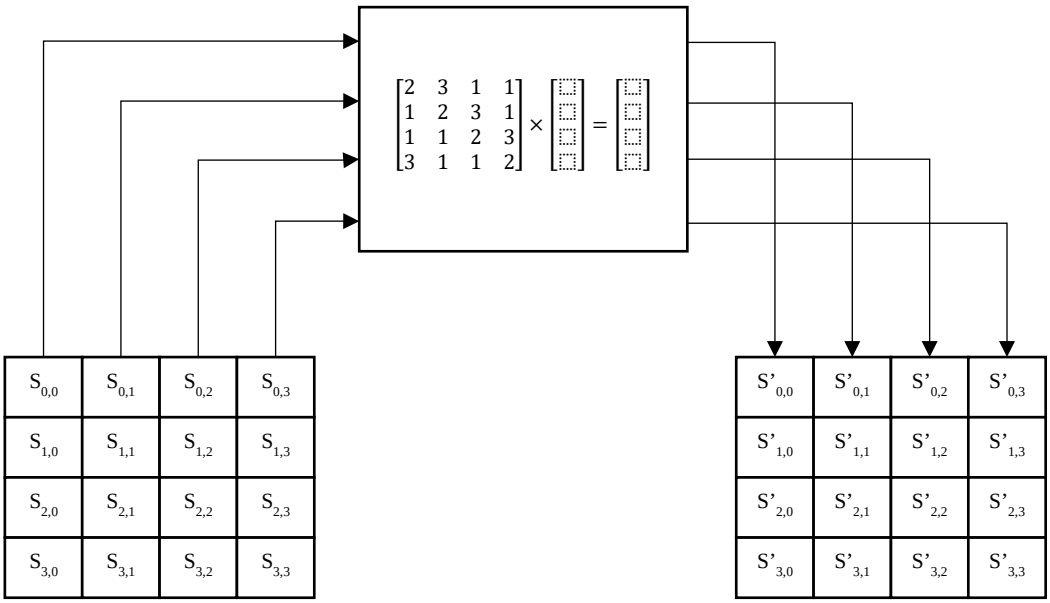


Figura 14 Transformação da mistura das colunas

2.4.2. CRIPTOANÁLISE

Como o AES se tornou o algoritmo de chave simétrica mais usado e conhecido em todo o mundo, ele também foi muito testado pelos criptoanalíticos. Existem inúmeros documentos e artigos pela Internet com diversos tipos de ataques, alguns com mais sucesso do que outros. A seguir vão ser apresentados alguns ataques, alguns com mais sucesso do que outros, mas nenhum sem derrubar o AES com sucesso, pois precisam de uma quantidade absurda de dados, tempo e memória, sendo só de interesse teórico.

Ataque de força bruta, criptoanálise linear e criptoanálise diferencial

O AES já exige, no mínimo, 128 bits de chave, para tornar impraticável o ataque de força bruta a este tipo de cifras. Por exemplo, ao usar uma cifra com 128 bits de chave é necessário realizar o processo de encriptação mais a comparação entre a saída e o texto cifrado original 2^{128} vezes. Isto faz com que seja necessário mais do que o tempo da existência do universo para quebrar a cifra, usando o poder computacional atual. Contudo, este ataque pode ser utilizado em conjunto com outro tipo de ataque para descobrir uma parte da chave que faltava descobrir.

Para prevenir possíveis ataques da criptoanálise linear, os autores do algoritmo AES desenvolveram-no utilizando a estratégia de design *Wide Trail* [16]. A estratégia tem como objetivo limitar as correlações de bits entre rondas, diminuindo as expressões lineares com grande probabilidade de decifrarem a mensagem. Por este motivo e devido à grande quantidade de textos puros exigidos para realizar este ataque, acaba por este tipo de ataque não ser prático. Até aos dias de hoje, não foram desenvolvidas criptoanálises lineares para atacar o AES.

A estratégia de design *Wide Trail* também diminui os rastos diferenciais para apenas 4 rondas do AES. Como o AES completo tem mais do que 4 rondas é fácil provar que o AES é bastante resistente à criptoanálise diferencial.

Criptoanálise diferencial impossível

Enquanto a criptoanálise diferencial procura por diferenças que se propagam com grande probabilidade no processo da encriptação, a criptoanálise diferencial impossível explora diferenças que são impossíveis (probabilidade 0) de acontecer no processo de encriptação. Existem vários ataques contra o AES usando este tipo de criptoanálise. A seguir vão ser apresentados os ataques mais recentes com a respetiva complexidade de dados necessária para realizar o ataque e o respetivo sucesso.

Em Janeiro de 2010, *Zheng Yuan* propôs expressões para serem utilizadas em novos ataques de criptoanálise diferencial impossível, tentando assim melhorar a eficácia deste tipo de ataques [18]. Também estudou as propriedades da criptoanálise diferencial impossível quando usada no estudo de 5 rondas adjacentes. Todos estes dados novos que ele apresenta podem ser usados para realizar ataques ao AES de 128 bits com 7 rondas, AES de 192 bits com 7 a 9 rondas e o AES de 256 bits com 8 a 12 rondas. Por último, ele também fez alguns ataques realizando estes métodos contra a cifra AES em diversas configurações. Na tabela 18 estão apresentados os resultados que ele obteve numa forma resumida.

Tabela 18 Resultados dos ataques do *Zheng Yuan*

Tamanho da chave	Número de rondas	Complexidade de dados	Complexidade de tempo
AES-128	7	$2^{115,32}$	$2^{119,32}$
AES-192	7	$2^{109,67}$	$2^{154,67}$
	8	$2^{102,3}$	$2^{166,3}$
	9	$2^{115,89}$	$2^{180,89}$
	9	$2^{125,89}$	$2^{150,89}$
AES-256	11	$2^{122,4}$	$< 2^{254,4}$

Em Dezembro de 2010, *Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen* e *Mahmoud Modarres-Hashemi* apresentaram um ataque contra o AES de 128 bits com 7 rondas, onde ele é quebrável com uma complexidade de dados, de cerca, de 2^{106} textos puros escolhidos e uma complexidade de tempo de 2^{110} encriptações [17]. Este ataque é o ataque mais eficiente conhecido até esta altura contra o AES de 128 bits.

Os ataques anteriores são os ataques mais recentes contra o AES usando a criptoanálise diferencial impossível. Contudo, existem muitos outros antes da data de 2010. Na tabela 19 estão exibidos alguns dados de alguns ataques conhecidos anteriores a esta data e com a devida referência onde, se necessário, é possível ir consultar para uma leitura mais aprofundada sobre o respetivo ataque.

Tabela 19 Resultados de alguns ataques conhecidos com a criptoanálise diferencial impossível

Tamanho da chave	Número de rondas	Complexidade de dados	Complexidade de tempo	Referência
AES-128	7	$2^{115,5}$	2^{119}	19, 20
	7	$2^{112,2}$	$2^{117,2}$	21
AES-192	7	2^{92}	$2^{186,2}$	22
	7	$2^{115,5}$	2^{119}	20
	7	$2^{113,8}$	$2^{118,8}$	21
	7	2^{92}	2^{162}	20
	7	$2^{91,2}$	$2^{139,2}$	21
AES-256	7	$2^{92,5}$	$2^{250,5}$	22
	7	2^{92}	2^{163}	21
	7	$2^{115,5}$	2^{119}	20
	7	$2^{113,8}$	$2^{118,8}$	21
	8	$2^{116,5}$	$2^{247,5}$	20
	8	$2^{111,1}$	$2^{227,8}$	21
	8	$2^{89,1}$	$2^{229,7}$	21

Ataque *boomerang*

O ataque *boomerang* foi desenvolvido em 1999 pelo *Wagner* [55] e é uma extensão muito boa da criptoanálise diferencial, pois permite quebrar mais rondas do que o ataque diferencial pode fazer. Isto, porque a cifra é tratada como uma cascata de duas subcifras, usando diferenciais pequenos em cada subcifra. Estes diferenciais são combinados com um ataque adaptativo de texto puro ou texto cifrado escolhido para explorar propriedades da cifra que têm uma alta probabilidade.

Em 2004, *Alex Biryukov* desenvolveu este tipo de ataque contra o AES de 128 bits reduzido de 5 e 6 rondas [23]. O ataque de *Biryukov* contra o AES de 5 rondas tem uma complexidade de dados de 2^{39} e para o AES de 6 rondas tem uma complexidade de dados de 2^{71}. Em 2008, foi a vez de *Michael Gorski* e *Stefan Lucks* desenvolverem dois ataques de *boomerang* com chave relacionada contra o AES de 192 bits de 7 e 9 rondas [24]. Os ataques de chave relacionada aplicam a criptoanálise diferencial à cifra usando chaves diferentes, mas relacionadas e analisa as encriptações de modo a tentar extrair informações relevantes sobre essas chaves. As cifras com uma programação da chave fraca são vulneráveis a este tipo de

ataques. A ideia deste tipo de ataques foi dada pela primeira vez por *John Kelsey, Bruce Schneier* e *David Wagner* em 1997 [25]. A partir daí várias combinações de ataques diferenciais e chaves relacionadas foram criados. Voltando aos ataques de *Gorski* e *Lucks*, eles apresentaram dois ataques, onde usam 4 chaves relacionadas para atacar o AES de 192 bits de 7 rondas e outro usam 256 chaves relacionadas para atacar o AES de 192 bits com 9 rondas. Estes ataques têm uma complexidade de dados de 2^{18} e de tempo de $2^{67,5}$ para o primeiro ataque e 2^{67} e $2^{143,33}$ respetivamente para o segundo ataque. Este ataque foi o ataque mais eficaz desenvolvido até essa altura para as cifras AES de 192 bits com o mesmo número reduzido de rondas.

Em 2009, *Alex Biryukov* e *Dmitry Khovratovich* apresentaram dois ataques de chave relacionada contra o AES completo, um para o AES de 192 bits e outro para o AES de 256 bits [26]. Em ambos os ataques, eles utilizam 4 chaves relacionadas. Para o AES de 256 bits, eles apresentaram o primeiro ataque de recuperação da chave que funciona para todas as chaves e tem uma complexidade de dados e do tempo de $2^{99,5}$. O segundo ataque deles é o primeiro ataque contra o AES de 192 bits completo. Estes dois ataques são ataques *boomerang*, eles tentam encontrar colisões locais e utilizam as técnicas *boomerang* para ganhar rondas gratuitas pelo meio.

Em 2011, *Jiqiang Lu* criou uma variante do ataque *boomerang*, onde usa a criptoanálise diferencial impossível em conjunto com o ataque *boomerang*, dando-lhe o nome de ataque *boomerang* impossível [27]. Depois ainda criou uma extensão deste ataque para ser utilizado num ataque de chave relacionada. Ele aplicou o ataque *boomerang* impossível (sem a extensão) ao AES de 128 bits com 6 rondas e aos AES de 192 e 256 bits com 7 rondas. Depois, com o ataque *boomerang* impossível com duas chaves relacionadas atacou o AES de 192 bits com 8 rondas e o AES de 256 bits com 9 rondas. Com estes ataques novos que ele criou, ele obteve os resultados que estão na tabela 20. De salientar, que os resultados que ele obteve com os ataques de chave relacionada foram, até à data do lançamento do artigo, os melhores resultados.

Tabela 20 Resultados que *Lu* obteve com os seus ataques

Cifra	Chaves	Técnica de ataque	Rondas	Complexidade de dados	Complexidade de tempo
AES-128	1	*Boomerang* impossível	6	$2^{112,2}$	$2^{112,3}$
AES-192	1	*Boomerang* impossível	7	$2^{112,5}$	$2^{186,3}$
	2	*Boomerang* impossível com chaves relacionadas	8	$2^{122,4}$	$2^{167,7}$
AES-256	1	*Boomerang* impossível	7	$2^{112,8}$	$2^{186,9}$
	2	*Boomerang* impossível com chaves relacionadas	9	2^{123}	$2^{239,9}$

Ataque *meet-in-the-middle*

Apesar de o ataque *meet-in-the-middle* não requerer muita complexidade de dados em relação a outras criptoanálises (necessita de poucos textos puros), a não linearidade entre as subchaves do AES mostra-se como um obstáculo para realizar este ataque. O primeiro ataque *meet-in-the-middle* criado para o AES foi o de *Hüseyin Demirci* e *Ali Aydın Selçuk* em 2008 contra o AES de 192 bits e AES de 256 bits [28]. Este ataque é direcionado para versões reduzidas do AES: 7 rondas para o AES de 192 bits e 7 e 8 rondas para o AES de 256 bits. Contudo, a complexidade necessária para realizar este ataque é elevada, como se pode verificar na tabela 21.

Tabela 21 Resultados do ataque de *Demirci* e *Selçuk*

Cifra	Rondas	Complexidade de dados	Complexidade de memória	Complexidade de tempo	Complexidade de pré-computação
AES-192	7	2^{32}	2^{206}	2^{72}	2^{208}
AES-256	7	2^{32}	2^{206}	2^{72}	2^{208}
	8	2^{32}	2^{206}	2^{200}	2^{208}

Em 2009, *Hüseyin Demirci*, *İhsan Taşkın*, *Mustafa Çoban* e *Adnan Baysal* melhoraram este ataque. Eles descobriram que uma parte da função interna da encriptação do AES pode ser expressa em poucas constantes em certas condições [29]. Eles utilizaram esta vulnerabilidade para explorar ataques *meet-in-the-middle* contra o AES de 128 e 192 bits com 7 rondas e o AES de 256 bits com 8 rondas. Os resultados que eles obtiveram para as diferentes versões do AES estão descritos na tabela 22.

Tabela 22 Resultados do ataque de *Demirci, Taşkın, Çoban* e *Baysal*

Cifra	Rondas	Complexidade de dados	Complexidade de memória	Complexidade de tempo	Complexidade de pré-computação
AES-128	7	2^{80}	2^{122}	2^{113}	2^{123}
AES-192	7	2^{80}	2^{122}	2^{113}	2^{123}
AES-256	7	2^{80}	2^{122}	2^{113}	2^{123}
	8	2^{80}	2^{123}	2^{241}	2^{124}

Mais tarde, em 2011, *Yongzhuang Wei*, *Jiqiang Lu* e *Yupu Hu* apresentaram um novo ataque *meet-in-the-middle* para a cifra AES de 192 bits com 8 rondas [30]. Eles basicamente juntaram dois trabalhos/ataques anteriormente realizados para obterem um novo ataque com maior sucesso. Eles pegaram no ataque *meet-in-the-middle* de *Demirci* e *Selçuk* de 2008, alteraram-no para atacar o AES de 192 bits com 8 rondas e juntaram o ataque de observação da programação da chave de *Dunkelman*, *Keller* e *Shamir* [31] para tornar o ataque mais eficiente. Com isto, conseguiram criar o ataque com melhores resultados criptoanalíticos contra o AES de 192 bits com 8 rondas até aquela altura. Eles conseguiram diminuir em muito a complexidade de dados necessária para realizar o ataque, tendo obtido os seguintes resultados gerais:

- **Complexidade de dados:** 2^{41};
- **Complexidade de memória:** 2^{190};
- **Complexidade de tempo:** $2^{187,63}$;
- **Complexidade de pré-computação:** $2^{187,63}$.

Em 2013, foi a vez de *Leibo Li*, *Keting Jia* e *Xiaoyun Wang* darem mais um passo importante nos ataques de *meet-in-the-middle* contra o AES [32]. Eles estudaram um ataque *meet-in-*

the-middle de recuperação da chave contra o AES de 192 bits com 9 rondas e usando o modelo de chave única. Eles com uma técnica nova e com o diferenciador de 5 rondas conseguiram diminuir em muito a complexidade de memória necessária para fazer o ataque. As complexidades de dados, tempo e memória do ataque são: 2^{121} textos puros escolhidos, 2^{185} encriptações e 2^{185} memórias de 128 bits, respetivamente.

Por último, em 2014, *Patrick Derbez* e *Pierre-Alain Fouque* [33] reviram o ataque *meet-in-the-middle* de *Demirci* e *Selçuk* de 2008. Eles encontraram uma maneira de modelar automaticamente as cifras de blocos SPN (*Substitution-Permutation Network*) e ataques *meet-in-the-middle*, permitindo realizar uma busca exaustiva a este tipo de ataques. Esta busca utiliza ferramentas desenvolvidas por *Bouillaguet*, *Derbez* e *Fouque* [34] como uma sub-rotina para resolver sistemas específicos. Também utilizaram algumas ideias do trabalho de *Dunkelman*, *Keller* e *Shamir* [31], mais precisamente a relação de compromisso de tempo/memória, e melhoraram-na. Depois, pegaram nos ataques melhorados de *Derbez*, *Fouque* e *Jean* [35] e aplicaram as técnicas que eles desenvolveram para melhorarem esses ataques contra as cifras AES de 192 e 256 bits com 8 rondas. Na tabela 23 estão os resultados que eles obtiveram.

Tabela 23 Resultados do ataque de *Derbez* e *Fouque*

Cifra	Rondas	Complexidade de dados	Complexidade de memória	Complexidade de tempo
AES-128	6	2^{8}	$2^{106,17}$	$2^{106,17}$
AES-192	7	2^{32}	$2^{129,67}$	$2^{129,67}$
	8	$2^{104,83}$	$2^{138,17}$	2^{140}
	8	2^{113}	2^{130}	2^{140}
AES-256	8	$2^{102,83}$	$2^{140,17}$	2^{156}
	8	2^{113}	2^{130}	2^{156}

Ataque *biclique*

O ataque *biclique* é uma variante do ataque *meet-in-the-middle*, ele utiliza uma estrutura de grafo bipartido completo para aumentar o número de rondas que podem ser atacadas pelo ataque *meet-in-the-middle*. Esta variante foi sugerida pela primeira vez por *Khovratovich*, *Rechberger* e *Savelieva* para atacar as cifras *Skein-512* e SHA-2 em 2012 [36]. Contudo, foi em 2011, que *Bogdanov*, *Khovratovich* e *Rechberger* mostraram como aplicar o conceito às

cifras de bloco de chave secreta quando aplicaram o ataque ao AES [37]. Eles conseguiram obter resultados muito significativos, incluindo o primeiro método de recuperação da chave para todas as versões do AES completo mais rápido do que o ataque de força bruta. O ataque deles teve as seguintes inovações:

- Foi o primeiro método de recuperação da chave para o AES de 128 bits completo, com uma complexidade computacional de $2^{126,1}$;
- Foi o primeiro método de recuperação da chave para o AES de 192 bits completo, com uma complexidade computacional de $2^{189,7}$;
- Foi o primeiro método de recuperação da chave para o AES de 256 bits completo, com uma complexidade computacional de $2^{254,4}$;
- Tem métodos de recuperação da chave, com menor complexidade, para as versões do AES com rondas reduzidas que não foram consideradas antes, incluindo a criptoanálise da versão de 8 rondas do AES de 128 bits, com uma complexidade computacional de $2^{124,9}$.

Na tabela 24 estão os resultados que *Bogdanov, Khovratovich* e *Rechberger* obtiveram com o ataque *biclique* para recuperar a chave nas diferentes versões do AES.

Tabela 24 Resultados de *Bogdanov, Khovratovich* e *Rechberger* com o ataque *biclique*

Cifra	**Rondas**	**Complexidade de dados**	**Complexidade de memória**	**Complexidade de tempo**	**Comprimento do *biclique* em rondas**
AES-128	8	$2^{126,33}$	2^{102}	$2^{124,97}$	5
	8	2^{127}	2^{32}	$2^{125,64}$	5
	8	2^{88}	2^{8}	$2^{125,34}$	3
	10	2^{88}	2^{8}	$2^{126,18}$	3
AES-192	9	2^{80}	2^{8}	$2^{188,8}$	4
	12	2^{80}	2^{8}	$2^{180,74}$	4
AES-256	9	2^{120}	2^{8}	$2^{253,1}$	6
	9	2^{120}	2^{8}	$2^{251,92}$	4
	14	2^{40}	2^{8}	$2^{254,42}$	4

Em 2015, *Biaoshuai Tao* e *Hongjun Wu* melhoraram este ataque [38]. Para conseguir isto, eles aumentaram o tamanho do *biclique* para $2^{16} \times 2^{8}$ e $2^{16} \times 2^{16}$ e utilizaram a técnica *Sieve-In-the-Middle* (SIM) para acelerar ainda mais a computação e armazenar as possíveis transições numa *superbox* (uma tabela de tamanho 2^{32}), mas com um custo extra de acesso a grandes tabelas. Este ataque tem as melhores complexidades de tempo para atacar as 3 versões do AES do que qualquer outro ataque. Os resultados que eles obtiveram encontram-se na tabela 25.

Tabela 25 Resultados de *Tao* e *Wu* com o ataque *biclique* melhorado

Cifra	Complexidade de dados	Complexidade de memória sem SIM	Complexidade de tempo sem SIM	Complexidade de memória com SIM	Complexidade de tempo com SIM
AES-128	2^{56}	$2^{22,07}$	$2^{126,13}$	2^{64}	$2^{125,99}$
	2^{72}	$2^{26,14}$	$2^{126,01}$	2^{64}	$2^{125,87}$
AES-192	2^{48}	$2^{22,27}$	$2^{189,91}$	2^{64}	$2^{189,76}$
AES-256	2^{40}	$2^{22,61}$	$2^{254,27}$	2^{64}	$2^{254,18}$

2.5. *TINY ENCRYPTION ALGORITHM* (TEA)

O *Tiny Encryption Algorithm* (TEA) é um algoritmo de chave simétrica criado por *David Wheeler* e *Roger Needham* da Universidade de *Cambridge* do Reino Unido e publicado em 1994 [51]. O TEA foi concebido para ser extremamente pequeno e ocupa muito pouco espaço em memória. Isto foi conseguido, fazendo o algoritmo com operações muito básicas e simples e a segurança é conseguida através da repetição destas operações muitas vezes. Como o algoritmo é construído com base em operações muito simples, ele é considerado um algoritmo muito rápido. Estas propriedades fazem com que o algoritmo seja muito escolhido para ser usado em implementações com *software* ou *hardware* fraco. O algoritmo pode substituir o algoritmo DES em muitas situações.

O TEA é um tipo de cifra de *Feistel*, que utiliza grupos de operações algébricas ortogonais mistas. A especificação do TEA sugere que ele tenha 32 rondas (64 rondas de *Feistel*) para cada bloco de dados de 64 bits. Cada ronda do TEA tem 2 operações de *Feistel*, adições, operações lógicas XOR e operações de deslocamento esquerda e direita. A chave está fixada em 128 bits (K) para ser resistente a ataques de força bruta e é dividida em 4 blocos de 32

bits (K[0], K[1], K[2], K[3]). As chaves K[0] e K[1] são usadas nas rondas ímpares e as chaves K[2] e K[3] são usadas nas rondas pares. O tamanho do bloco de dados de entrada é de 64 bits, que deve ser dividido em duas palavras de 32 bits.

2.5.1. O PROCESSO DE ENCRIPTAÇÃO E DESENCRIPTAÇÃO DO TEA

Na figura 15 está representada a rotina de encriptação do TEA. As entradas do algoritmo de encriptação são: um bloco de texto puro (P) e uma chave (K). O bloco de texto puro é dividido em duas partes: P = (Esquerda [0], Direita [0]), resultando no final um texto cifrado também dividido em duas partes: C = (Esquerda [64], Direita [64]). Cada parte do texto puro é usada para encriptar a outra parte do texto puro ao longo de 64 rondas de processamento e, em seguida, combinam-se para produzir o bloco de texto cifrado. Cada ronda 'i' tem como entradas: Esquerda (i-1) e Direita (i-1), derivadas da ronda anterior, e uma subchave (K[i]) derivada da chave de 128 bits (K). A subchave e a chave de 128 bits são diferentes uma da outra. A programação da chave é, simplesmente, um OR (porta lógica OU) das palavras da chave com um valor deslocado do último estado de cada uma das palavras do bloco. Esta operação faz com que todos os bits da chave e dos dados sejam misturados repetidamente. O TEA utiliza um valor delta na especificação, que é definido como $\sqrt{5} - 1 \times 2^{31} = 9E3779B9_h$, que vem da proporção áurea. Um múltiplo diferente delta é usado em cada ronda para evitar as explorações de simetria com base nas operações de *Feistel*. O processo de desencriptação é essencialmente o mesmo ao processo de encriptação, só que a entrada do algoritmo é o texto cifrado e as subchaves são usadas pela ordem inversa.

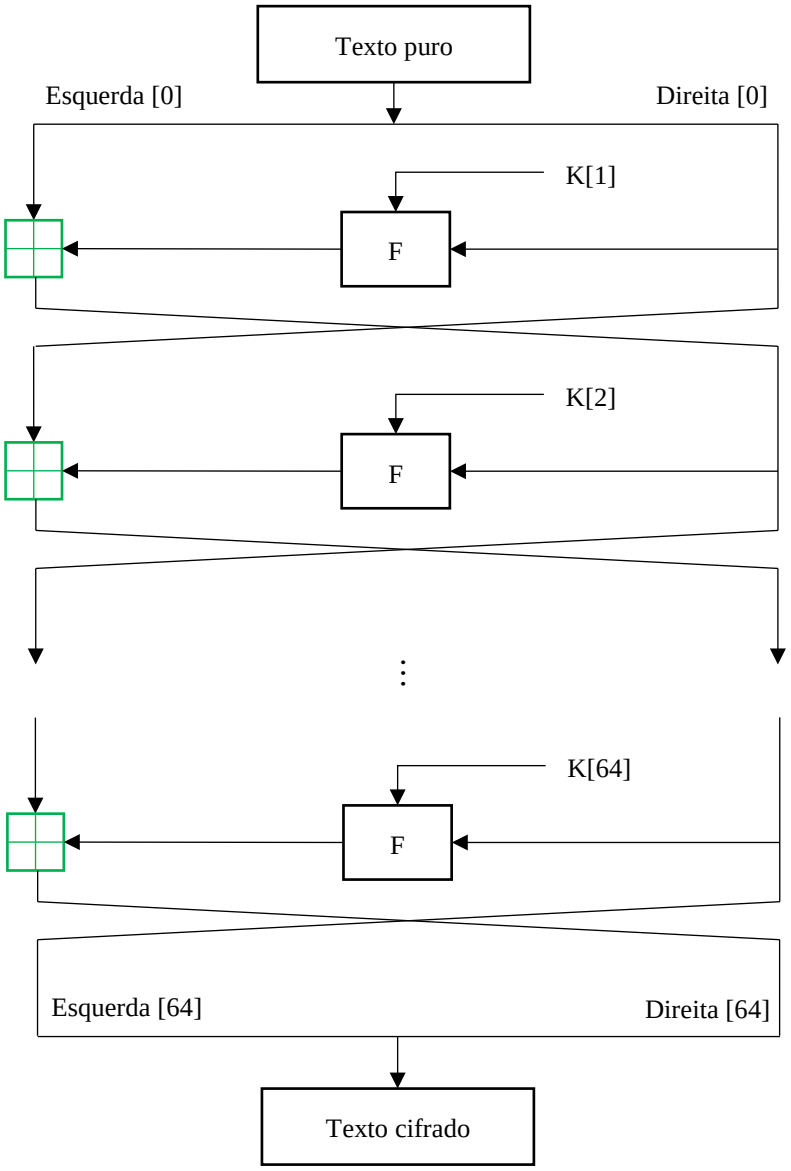


Figura 15 A estrutura da rotina de encriptação do TEA

A figura 16 mostra os detalhes internos de um ciclo TEA. A função de ronda (F) consiste na adição da chave, operações lógicas XOR e operações de deslocamento esquerda e direita. Pode-se descrever as saídas de um 'i' ciclo (Esquerda [i+1] e Direita [i+1]) com as suas entradas (Esquerda [i] e Direita [i]) usando as fórmulas 25 e 26:

$$Esquerda\,[i+1] = Esquerda\,[i] \boxplus F(Direita\,[i], K[0,1], delta[i]) \quad (25)$$

$$Direita\,[i+1] = Direita\,[i] \boxplus F(Direita\,[i+1], K[2,3], delta[i]) \quad (26)$$

$$delta[i] = \frac{i+1}{2} * delta \quad (27)$$

A função de ronda (F) é definida pela fórmula 28. Ela tem a mesma estrutura geral para cada ronda, só que cada ronda tem a sua subchave de ronda (*K[i]*).

$$F(M, K[j,k], delta[i]) = \left((M \ll 4) \boxplus K[j]\right) \oplus (M \boxplus delta[i]) \oplus \left((M \gg 5) \boxplus K[k]\right) \quad (28)$$

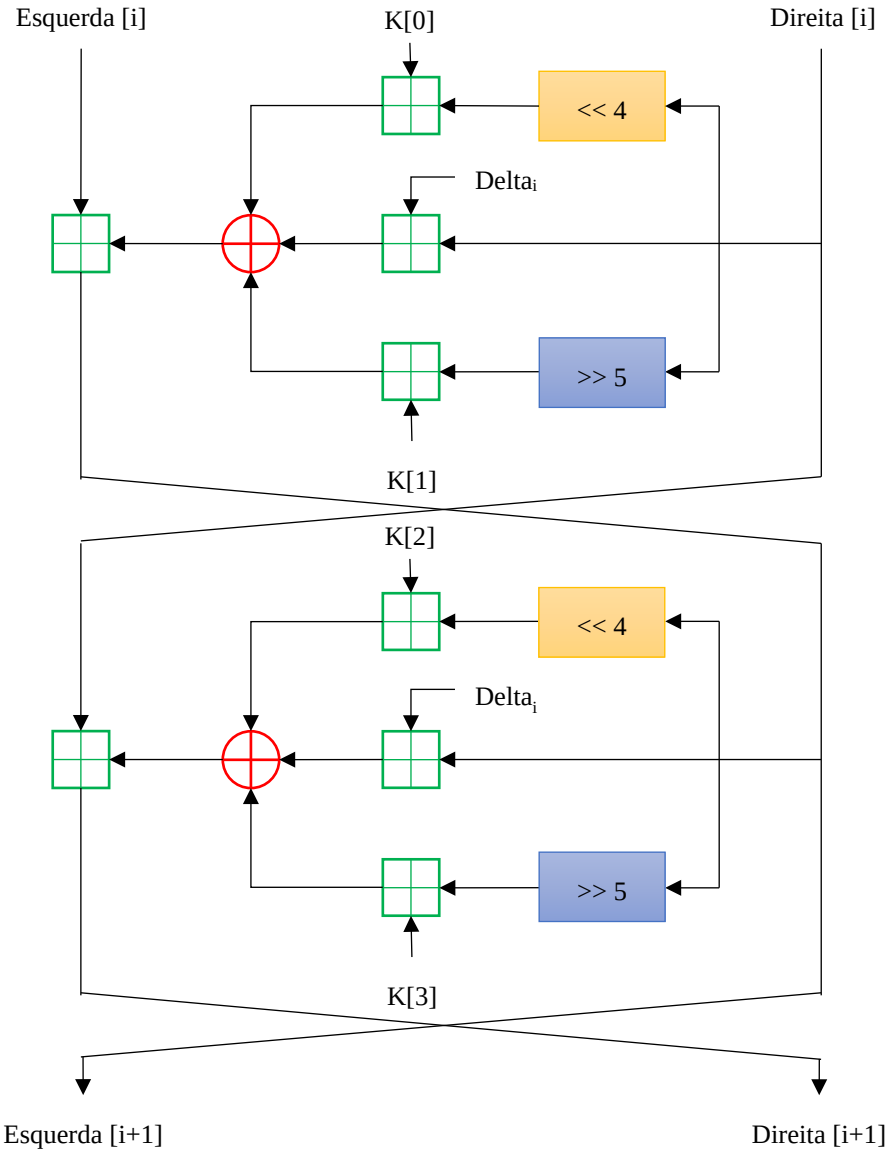


Figura 16 Detalhes internos de um ciclo TEA (2 rondas de *Feistel*)

2.5.2. CRIPTOANÁLISE

Tem havido muitas tentativas para encontrar fraquezas no TEA e há muitos ataques que fizeram o TEA quebrável. O principal problema encontrado nele é que ele usa chaves equivalentes no seu algoritmo, enfraquecendo assim a eficácia do tamanho da chave e, por

isso, requer apenas uma complexidade de 2^{32} para quebrar a chave, usando um ataque de chave relacionada. Esta complexidade é muito menor do que a complexidade necessária para realizar o ataque de força bruta, que é de 2^{128}. Em 1997, *John Kelsey* e *Bruce Schneier* apresentaram um artigo, onde usam a técnica de ataque de chave relacionada contra várias cifras, entre elas o TEA [45]. Eles apresentaram 4 ataques contra o TEA, em que o melhor ataque combina o ataque da rotação da chave de *Biham* e *Biryukov* [43] e a criptoanálise diferencial com chave relacionada de *Kelsey*, *Schneier* e *Wagner* [44], precisando, de apenas, 2^{23} de textos puros escolhidos, uma consulta de chave relacionada e cerca de 2^{32} computações para recuperar toda a chave.

Criptoanálise diferencial impossível

Em 2002, *Dukjae Moon*, *Kyungdeok Hwang*, *Wonil Lee*, *Sangjin Lee* e *Jongin Lim* desenvolveram um ataque para versões reduzidas do TEA, tendo por base a criptoanálise diferencial impossível [46]. Eles conseguiram obter os melhores resultados até essa altura, menos para o ataque de chave relacionada. Nesse artigo, eles mostraram uma construção de uma caraterística impossível para o TEA de 10 rondas e depois usaram essa caraterística para atacar o TEA de 11 rondas com $2^{52,5}$ textos puros escolhidos e 2^{84} encriptações.

Em 2012, *Chen*, *Wang* e *Preneel* apresentaram um ataque [47], onde melhoram os resultados do ataque de 2002. Eles identificaram diferenciais impossíveis da versão de 13 rondas do TEA e utilizaram-nos para atacar a versão de 17 rondas. Este ataque precisa de 2^{57} textos puros escolhidos e $2^{106,6}$ encriptações. Apesar de este ataque melhorar consideravelmente o ataque diferencial impossível de 2002, não é melhor do que o ataque de criptoanálise linear com correlação zero e com redução da complexidade dos dados de *Bogdanov* e *Wang* de 2012 [48].

Criptoanálise diferencial

O TEA é muito resistente à criptoanálise diferencial porque tem uma difusão completa, onde um bit diferente no texto original vai causar, aproximadamente, 32 bits diferentes no texto cifrado. Contudo, em 2004, *Seokhie Hong*, *Deukjo Hong*, *Youngdai Ko*, *Donghoon Chang*, *Wonil Lee* e *Sangjin Lee* apresentaram um ataque utilizando a criptoanálise diferencial truncada [49]. Enquanto a criptoanálise diferencial analisa a diferença total entre os dois textos, a variante truncada só considera as diferenças que são parcialmente determinadas, ou

seja, o ataque faz previsões de apenas uma parte dos bits em vez do bloco completo. O ataque deles ataca o TEA com 17 rondas, necessitando de $2^{123,73}$ encriptações. Este resultado foi melhor do que o ataque de criptoanálise diferencial impossível de 2002.

Criptoanálise linear com correlação zero

A criptoanálise linear com correlação zero é uma nova técnica de recuperação da chave para as cifras de bloco propostas pelo *Bogdanov* e *Rijmen* em 2011 [50]. Baseia-se em aproximações lineares com probabilidade de exatamente ½, que corresponde à correlação zero. Por outras palavras, a criptoanálise linear com correlação zero é uma aproximação linear que descreve uma cifra de bloco com uma correlação de exatamente zero. Para construir a aproximação linear é escolhido padrões de entrada e de saída, de tal modo que não haja nenhuma fuga linear com correlação diferente de zero para a correlação linear. Algumas cifras de bloco acabam por ter várias aproximações lineares com correlação zero para cada chave ao longo de um número considerável de rondas. Pode-se dizer que a criptoanálise linear com correlação zero está para a criptoanálise linear como a criptoanálise impossível está para a criptoanálise diferencial. No entanto, têm muitas diferenças técnicas e, por vezes, a criptoanálise linear com correlação zero resulta em ataques mais fortes.

Em 2012, *Bogdanov* e *Wang*, propuseram uma técnica estatística que influencia, de forma significativa, a redução da complexidade dos dados, usando um elevado número de aproximações lineares com correlação zero disponíveis [48]. Eles também identificaram aproximações lineares com correlação zero para o TEA com 14 e 15 rondas, resultando num ataque de recuperação da chave para o TEA com 21 rondas. Este ataque precisa de $2^{62,62}$ textos puros conhecidos e $2^{121,52}$ de complexidade de tempo e tem uma probabilidade de sucesso de 84,6%. Eles também propõem um ataque, se todos os 2^{64} textos puros estiverem disponíveis ao atacante, contra o TEA com 23 rondas, com uma complexidade de tempo de $2^{119,64}$ e uma probabilidade de sucesso de 100%. Este trabalho demonstra que os requisitos de complexidade de dados (textos puros conhecidos para este caso) proibitivos não são inerentes à criptoanálise linear com correlação zero e podem ser superados. Além disso, os resultados deles sugerem que a criptoanálise linear com correlação zero pode quebrar mais rondas do que a criptoanálise diferencial impossível.

2.6. *eXTENDED* TEA (XTEA)

Depois de serem descobertas e documentados algumas fraquezas e vulnerabilidades do TEA, *Wheeler* e *Needham*, os criadores do TEA, decidiram apresentar uma nova implementação do TEA, chamando-lhe extensão do TEA ou, em inglês, *eXtended TEA* (XTEA). O XTEA foi apresentado em 1997 [52], três anos após o TEA ser apresentado. O XTEA partilha muitas das caraterísticas do TEA: usa operações aritméticas e lógicas, tem o mesmo bloco de 64 bits, a mesma chave de 128 bits e as mesmas 64 rondas de *Feistel* ou 32 rondas do XTEA. As maiores vulnerabilidades do TEA foram descobertas usando os ataques diferenciais de chave relacionada. Por isso, os autores com o XTEA tentaram corrigir as vulnerabilidades, melhorando alguns aspetos do algoritmo. Uma das grandes mudanças do algoritmo é relacionada precisamente com a introdução das subchaves, elas são adicionadas mais lentamente. Além disso, as subchaves são selecionadas usando a variável "sum" com dois bits. Também foi introduzido um deslocamento de 11 à programação da chave para ajudar a criar uma sequência irregular das subchaves. As outras alterações introduzidas no XTEA é um rearranjo da estrutura do algoritmo, como se pode ver na figura 17. Em vez da colocação definida das subchaves, agora as subchaves são introduzidas como subchave A e subchave B. Pode-se descrever as saídas de um 'i' ciclo (Esquerda [i+1] e Direita [i+1]) com as suas entradas (Esquerda [i] e Direita [i]) usando as fórmulas 29 e 30:

$$Esquerda\,[i+1] = Esquerda\,[i] \boxplus F(Direita\,[i], K[2\mathrm{i}-1], delta[i-1]) \tag{29}$$

$$Direita\,[i+1] = Direita\,[i] \boxplus F(esquerda\,[i+1], K[2i], delta[i]) \tag{30}$$

$$delta[i] = \frac{i+1}{2} * delta \tag{31}$$

A função de ronda (F) é definida pela fórmula 32:

$$F(M, K[*], delta[**]) = \big((M \ll 4) \oplus (M \gg 5)\big) \boxplus M \oplus delta[**] \boxplus K[*] \tag{32}$$

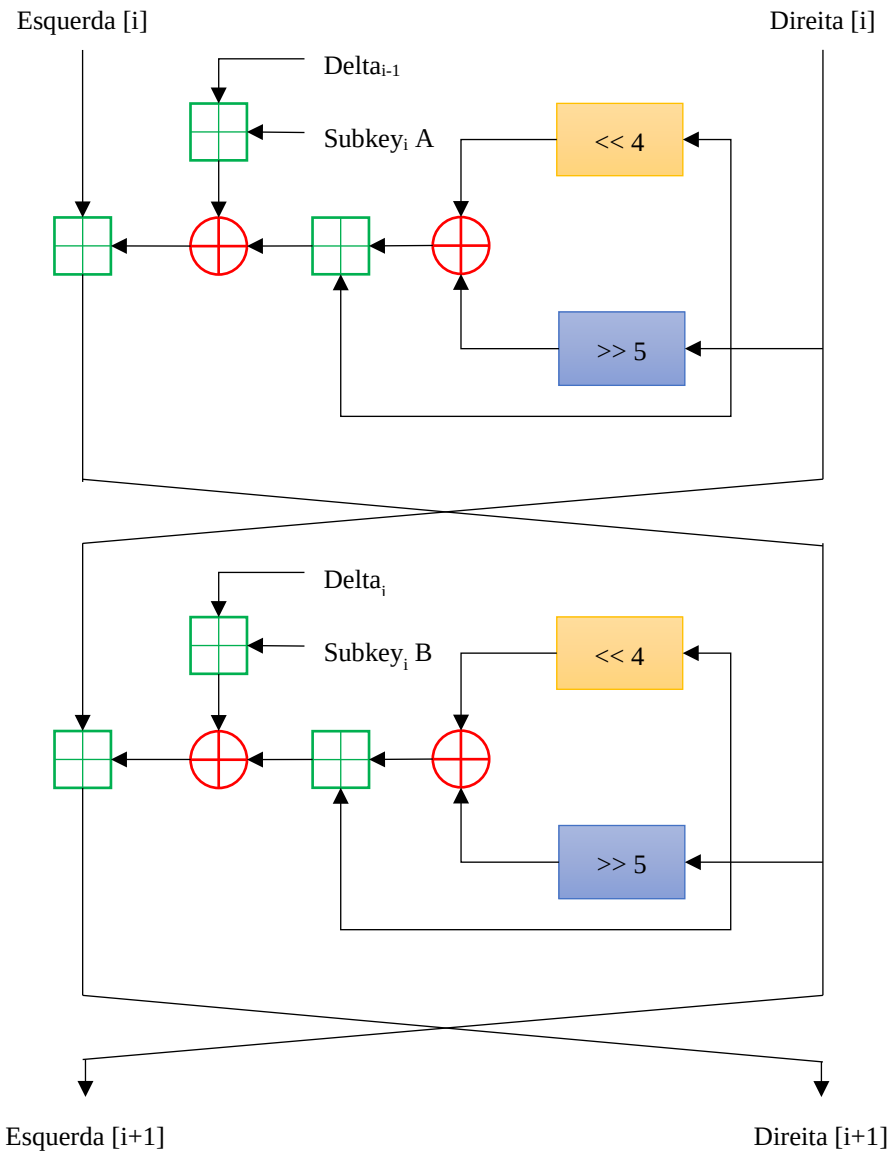


Figura 17 Detalhes internos de um ciclo XTEA (2 rondas de *Feistel*)

2.6.1. CRIPTOANÁLISE

O XTEA é uma cifra de bloco com uma estrutura muito simples, mas ainda não foram criados ataques para a versão completa da cifra. Contudo, foram publicados vários ataques contra a cifra XTEA reduzida. A seguir são apresentados os resultados de alguns deles.

Em 2002, *Dukjae Moon*, *Kyungdeok Hwang*, *Wonil Lee*, *Sangjin Lee* e *Jongin Lim* mostraram como construir uma caraterística impossível para o XTEA com 12 rondas [46].

Depois utilizam essa caraterística impossível que construíram para atacar o XTEA com 14 rondas, com $2^{62,5}$ textos puros escolhidos e 2^{85} encriptações.

Em 2004, *Seokhie Hong, Deukjo Hong, Youngdai Ko, Donghoon Chang, Wonil Lee* e *Sangjin Lee* apresentaram um ataque utilizando a criptoanálise diferencial truncada [49]. O ataque deles ataca o XTEA com 23 rondas, necessitando de $2^{120,65}$ encriptações. Também em 2004, *Youngdai Ko, Seokhie Hong, Wonil Lee, Sangjin Lee* e *Ju-Sung Kang* apresentaram um ataque diferencial truncado de chave relacionada contra o XTEA [54]. O ataque consegue atacar o XTEA com 27 rondas, com uma taxa de sucesso de 96,9% e requer cerca de $2^{20,5}$ textos puros escolhidos e $2^{115,15}$ encriptações.

Em 2006, *Eunjin Lee, Deukjo Hong, Donghoon Chang, Seokhie Hong* e *Jongin Lim* apresentaram uma classe de chaves fracas que faz o XTEA com 34 rondas vulnerável a um ataque retângulo de chave relacionada [56]. O ataque retângulo é uma variante do ataque *boomerang*, só que o ataque retângulo utiliza textos puros escolhidos para realizar os ataques e o ataque *boomerang* utiliza textos cifrados escolhidos. O número de chaves fracas que eles apresentam é de cerca de $2^{108,21}$ e o ataque requer 2^{62} textos puros escolhidos e $2^{31,94}$ encriptações.

Em 2011, *Sekar, Mouha, Velichkov* e *Preneel* apresentaram uns ataques *meet-in-the-middle* contra o XTEA para as versões de 7, 15 e 23 rondas [53]. Para as 23 rondas, o ataque deles necessita, de apenas, 18 textos puros conhecidos e uma complexidade de tempo de 2^{117} para testar as chaves, com uma probabilidade de sucesso de $1\text{-}2^{-1025}$. Este ataque era o que requeria menos tempo e menos dados do que qualquer outro ataque que saiu até essa altura para as 23 rondas ou mais. Os ataques para as 7 e 23 rondas usam o ataque *meet-in-the middle* simples, já para as 15 rondas o ataque *meet-in-the-middle* ataca as subchaves de ronda em vez dos textos intermédios. Na tabela 26 estão representados os resultados dos ataques que eles obtiveram.

Tabela 26 Resultados de *Sekar, Mouha, Velichkov* e *Preneel* com o ataque *meet-in-the-middle*

Rondas	Complexidade de dados	Complexidade de tempo	Probabilidade de sucesso
7	2	2^{95}	$1-2^{-33}$
15	3	2^{95}	$1-2^{-65}$
23	18	2^{117}	$1-2^{-1025}$

Em 2012, *Bogdanov* e *Wang* atacaram o XTEA recorrendo à criptoanálise linear com correlação zero [48]. Eles identificaram aproximações lineares com correlação zero para as 14 e 15 rondas do XTEA. Estas aproximações serviram para atacar o XTEA com 25 rondas, pelo método do ataque de recuperação da chave. Os resultados que eles obtiveram encontram-se na tabela 27.

Tabela 27 Resultados de *Bogdanov* e *Wamg* com o ataque de criptoanálise linear com correlação zero

Rondas	Complexidade de dados	Complexidade de memória	Complexidade de tempo	Probabilidade de sucesso
25	$2^{62,62}$	2^{30}	$2^{124,53}$	84,6%
27	2^{64}	Desprezível	$2^{120,71}$	100%

Também no ano de 2012, *Chen, Wang* e *Preneel* atacaram o XTEA com a criptoanálise diferencial impossível [47]. Eles propõem um método para identificar diferenciais impossíveis, tirando vantagem da fraca difusão da cifra. O método deles encontra diferenciais impossíveis para o XTEA com 14 rondas que resulta em ataque com diferenciais impossíveis contra o XTEA com 23 rondas. Este ataque melhora substancialmente os resultados obtidos em 2002 por *Dukjae Moon, Kyungdeok Hwang, Wonil Lee, Sangjin Lee* e *Jongin Lim* [46]. Os resultados que eles obtiveram encontram-se na tabela 28.

Tabela 28 Resultados de *Chen, Wang* e *Preneel* com o ataque de criptoanálise diferencial impossível

Rondas	Complexidade de dados	Complexidade de memória	Complexidade de tempo
23	$2^{62,3}$	Desprezível	$2^{114,9}$
23	2^{63}	2^{101}	$2^{105,6}$

2.7. *Corrected Block* TEA (XXTEA)

Em paralelo com o lançamento do XTEA, *Wheeler* e *Needham* lançaram o *Block* TEA. A grande inovação do *Block* TEA é que ele trabalha com tamanhos dos blocos variáveis desde que sejam múltiplos de 32 bits. O algoritmo aplica a função de ronda (F) do XTEA em cada palavra dos blocos e combina com o bloco seguinte. Isto é repetido em várias rondas, dependendo do tamanho do bloco (no mínimo 6 vezes). Uma das vantagens deste procedimento é que não é necessário trabalhar com os modos de operação[2] e, assim, a cifra pode ser aplicada diretamente na mensagem. A função de ronda (F) do *Block* TEA é praticamente a mesma do XTEA, como se pode ver na figura 18.

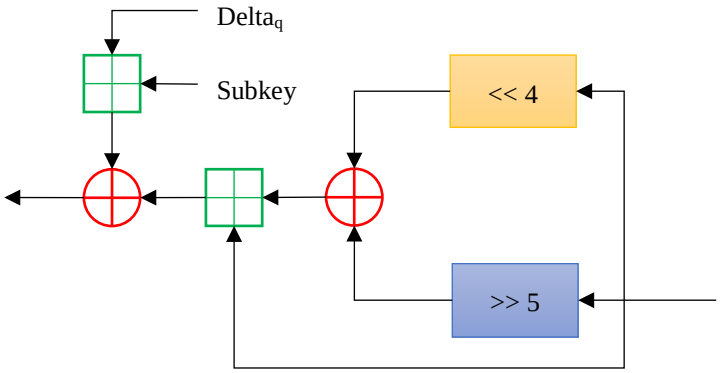


Figura 18 Função de ronda do *Block* TEA

Um ano mais tarde, em 1998, *Wheeler* e *Needham* sugeriram mais uma melhoria, aperfeiçoando a função de ronda do *Block* TEA e a versão é publicada com o nome *Corrected Block* TEA (XXTEA). O XXTEA como é derivado do *Block* TEA, ele tem algumas diferenças significativas em relação ao TEA e ao XTEA. O XXTEA trabalha com tamanhos de blocos múltiplos de 32 bits e com um tamanho da chave de 128 bits. Como o XXTEA não tem um limite de tamanho dos blocos, ele pode ser utilizado para encriptar a mensagem inteira sem a necessidade de ter um modo de operação. Contudo, o XXTEA pode trabalhar

[2] Os modos de operação são algoritmos usados nas cifras de bloco que fornecem à cifra confidencialidade e segurança. A cifra de bloco por si só encripta e desencripta um bloco, já um modo de operação tem como finalidade aplicar repetidamente essa operação de uma cifra para encriptar e desencriptar mais do que um bloco.

com um modo de operação para mensagens muito grandes e, assim, não precisam de ser carregadas inteiras para a memória. O número de rondas do XXTEA depende diretamente do tamanho dos blocos que se quer encriptar e é calculado pela fórmula 33, onde n é o tamanho do bloco em bits / 32.

$$p = 6 + 52 / n \tag{33}$$

Em contraste com o TEA e o XTEA, existem dois ciclos no processo de encriptação e desencriptação. O ciclo externo é o mesmo que o ciclo do TEA e do XTEA faz em cada ronda, enquanto o ciclo interno faz uma iteração para cada 32 bits do bloco (chamados segmentos). Em cada iteração é realizada operações para o segmento "Xr", envolvendo segmentos antes (Xr-1) e depois (Xr+1). Na figura 19 mostra a operação de encriptação de cada segmento do XXTEA.

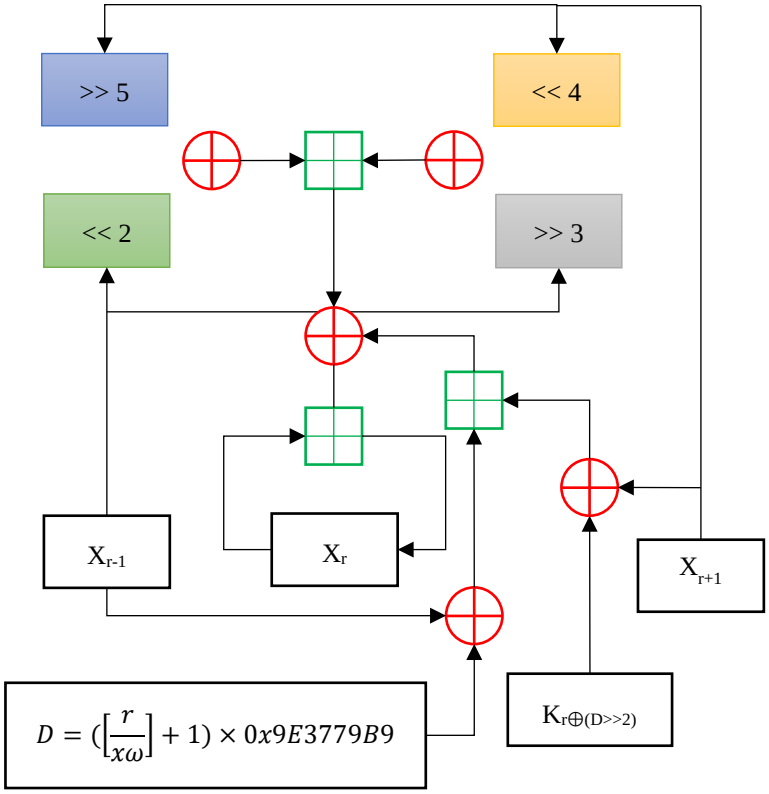


Figura 19 Uma ronda do XXTEA

2.7.1. CRIPTOANÁLISE

Em relação à cifra XXTEA, há um registo do ataque a ela e foi apresentado em 2010 por *Elias Yarrkov* [57]. Ele descreve um ataque de texto puro escolhido baseado na criptoanálise diferencial, usando cerca de 2^{59} consultas de texto puro escolhido e com pouco trabalho. O significado prático deste ataque é muito baixo, pois não há nenhuma maneira de o atacante obter até mesmo uma pequena fração de texto cifrado exigido, muito menos obter o texto puro escolhido.

3. DESEMPENHO DAS CIFRAS NO ARDUINO

A execução prática desta tese consiste na realização de testes de desempenho dos principais algoritmos de criptografia, de modo a analisar qual é a mais rápida no *hardware* Arduino e qual é a mais recomendada tendo em conta a relação desempenho/segurança.

Estes testes foram feitos num ambiente Arduino, fazendo uso das bibliotecas já desenvolvidas para ele. Pressupõe-se que as bibliotecas estão devidamente otimizadas para o Arduino e que, por isso, têm um desempenho por excelência. Obviamente, isto pode não corresponder à realidade. No futuro, certamente serão desenvolvidas outras bibliotecas mais otimizadas para o Arduino, atingindo assim um melhor desempenho. Tendo em conta isto, estes testes servem para fazer uma avaliação do desempenho atual das implementações existentes de encriptações e ter material para ser possível compará-las diretamente. Todos os códigos utilizados estão em anexo a este documento.

Foram testadas só as encriptações mais utilizadas e que ainda não tenham sido quebradas. Foi medido o tempo que demora a encriptar uma determinada palavra e a respetiva desencriptação. Foram utilizadas três palavras de entrada com diferentes tamanhos para testar os diferentes algoritmos que são: "*on*", "*password*" e "*0001:password:on*". Estas palavras servem para simular o envio de uma determinada informação para um determinado atuador: "0001" supõe-se que é a identificação do atuador na rede, "*password*" é a palavra-passe que é enviada e tem que corresponder à palavra-passe interna do atuador para a ordem ser executada e "*on*" é a ordem a ser executada pelo atuador.

O *hardware* utilizado para realizar o teste de desempenho das cifras foi uma placa Arduino Mega 2560, um computador e um cabo USB para ligar o computador ao Arduino através da porta série.

3.1. DESCRIÇÃO DO ARDUINO

A placa Arduino Mega 2560 é baseada no ATmega2560, tem 54 pinos de entradas e saídas digitais, em que 15 destes podem ser utilizados como saídas *Pulse-Width Modulation* (PWM). Tem também 16 entradas analógicas, 4 portas de comunicação série, um oscilador de cristal de 16 MHz, uma ligação USB, uma tomada de alimentação, um *In-Circuit Serial Programming* (ICSP) *header* e um botão de *reset*. Na tabela 29 tem-se o resumo das caraterísticas técnicas da placa Arduino Mega 2560 e na figura 20 uma imagem da placa.

Tabela 29 Resumo das caraterísticas técnicas do Arduino Mega 2560

Microcontrolador	ATmega2560
Tensão operacional	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limites)	6-20V
Pinos digitais de I/O	54, dos quais 15 fornece uma saída de PWM
Pinos de entrada analógica	16
Corrente DC por pino de I/O	20 mA
Corrente DC por pino de 3,3 V	50 mA
Memória flash	256 KB, dos quais 8 KB são usados pelo *bootloader*
SRAM	8 KB
EEPROM	4 KB
Velocidade do relógio	16 MHz
Comprimento	101,52 mm
Largura	53,3 mm
Peso	37 g

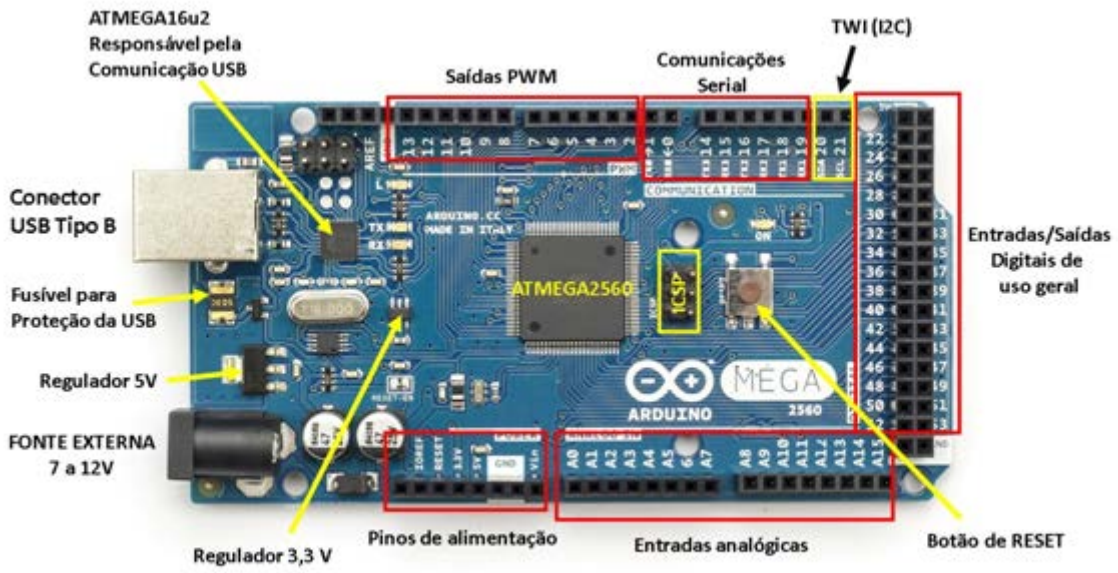


Figura 20 Arduino Mega 2560 [58]

3.1.1. MICROPROCESSADOR ATMEGA2560

O microprocessador utilizado pela placa do Arduino Mega 2560 é um microcontrolador da ATMEL, o ATMEGA 2560. Este microcontrolador é de 8 bits, com uma arquitetura *Reduced Instruction Set Computing* (RISC)[3] avançada. Ele tem 256 KB de memória flash (em que 8 KB são utilizados pelo *bootloader*), 8 KB de memória *Static Random Access Memory* (SRAM) e 4 KB de memória *Electrically Erasable Programmable Read-Only Memory* (EEPROM). Possui um multiplicador por *hardware* e vários periféricos que aumentam as possibilidades da plataforma Arduino, baseada em ATMEGA, como por exemplo, os 4 canais de comunicação série, 16 entradas analógicas e 15 saídas PWM. Tem ainda comunicações *Serial Peripheral Interface* (SPI), *Inter-Integrated Circuit* (I2C) e 6 pinos de interrupções externas.

[3] *Reduced Instruction Set Computing* (RISC) é um tipo de arquitetura de microprocessador que usa um pequeno conjunto, altamente otimizado de instruções, em vez de um conjunto mais especializado de instruções normalmente encontrado em outros tipos de arquiteturas.

3.1.2. ENTRADAS E SAÍDAS

A placa Arduino Mega 2560 tem 54 pinos de entradas e saídas digitais, que podem ser usados através das funções:

- ***pinMode():*** configura um determinado pino para se comportar como uma entrada ou uma saída;

- ***digitalWrite():*** escreve *HIGH* ou *LOW* para um determinado pino digital. Se o pino for configurado em *HIGH*, a tensão dele será definida para o valor de 5 V (ou 3,3 V para placas de 3,3 V), se for configurado em *LOW*, a tensão dele será definida para o valor de 0 V (tensão da terra);

- ***digitalRead():*** lê o valor de um determinado pino digital (*HIGH* ou *LOW*).

Os pinos de entradas e saídas digitais trabalham com tensões de 5 V e podem fornecer até 20 mA. Cada pino tem uma resistência interna de *pull-up* que pode ser ativada por *software*. Alguns destes pinos possuem funções especiais, como é explicado a seguir:

- **Para as comunicações série –** série: 0 (RX – *receive*) e 1 (TX – *transmit*), série 1: 19 (RX) e 18 (TX), série 2: 17 (RX) e 16 (TX), série 3: 15 (RX) e 14 (TX). Os pinos 0 e 1 estão ligados aos pinos do ATMEGA16U2, o responsável pela comunicação USB;

- **Para as interrupções externas –** 2 (interrupção 0), 3 (interrupção 1), 18 (interrupção 5), 19 (interrupção 4), 20 (interrupção 3) e 21 (interrupção 2). Estes pinos podem ser configurados para disparo da interrupção tanto para o flanco ascendente como para o flanco descendente ou em níveis lógicos de alto ou baixo, conforme a necessidade do projeto;

- **Para as saídas PWM –** os pinos 2 a 13 e 44 a 46 podem ser usados como saídas PWM. O sinal PWM tem 8 bits de resolução e é implementado com a função *analogWrite()*;

- **Para a comunicação SPI –** os pinos 50 (MISO – *Master In Slave* Out), 51 (MOSI – *Master Out Slave In*), 52 (SCK – *Serial Clock*) e 53 (SS – *Slave Select*);

- **Para a comunicação I2C –** os pinos (SDA – *Serial Data*) e 21 (SCL – *Serial Clock*).

O Arduino Mega 2560 também tem 16 entradas analógicas (pinos A0 e A15), onde cada uma tem uma resolução de 10 bits, isto é, 1024 valores diferentes. Por defeito, a tensão de referência é de 5 V para estes pinos, porém é possível mudar este valor através do pino AREF e a função *analogReference()*.

3.2. ARDUINO IDE

O Arduino IDE torna fácil escrever código e enviá-lo para o Arduino. O programa funciona em Windows, Mac OS X e Linux e é escrito em java. Ele tem um editor de código, permitindo ainda compilar e carregar programas para a placa com um único clique. A figura 21 mostra a interface deste programa.

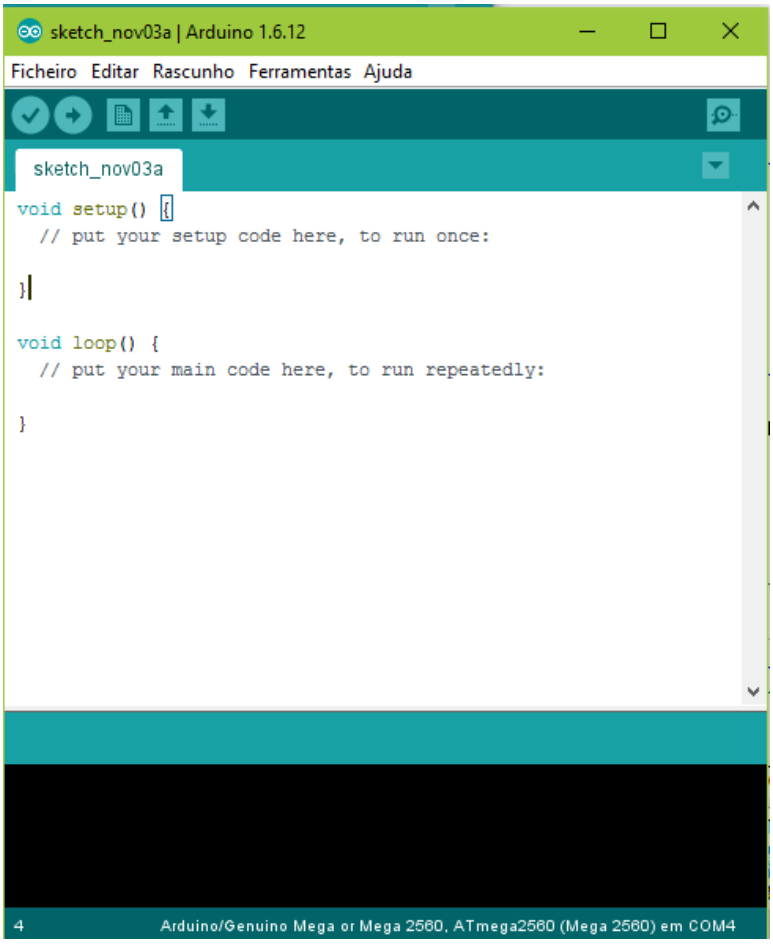


Figura 21 Programa IDE do Arduino

É possível programar em C/C++, permitindo assim criar com facilidade muitas operações de entrada e saída. O Arduino IDE usa o *GNU Compiler Collection* (GCC) para compilar o código. Para o programa criado funcionar precisa de ter apenas duas funções:

- ***setup()*:** uma função que é executada uma vez no inicio de um programa e que pode ser usada para iniciar a configuração;
- ***loop()*:** uma função que é sempre repetida até a placa ser desligada.

3.3. BIBLIOTECAS

As bibliotecas são ficheiros em C ou C++ que fornecem funcionalidades específicas para um programa, como por exemplo, a capacidade de escrever num display *Liquid Crystal Display* (LCD) ou de controlar a posição de um servomotor. As bibliotecas simplificam o desenvolvimento de programas, porque o código fica criado nessas bibliotecas e só é necessário integrar ao programa em desenvolvimento para que as funções dessas bibliotecas possam ser acedidas e utilizadas pelo programa. Na plataforma do Arduino estão disponíveis três tipos diferentes de bibliotecas: *Core* (biblioteca essencial), padrão e adicionais (de terceiros).

A biblioteca *Core* é uma biblioteca essencial e indispensável para o desenvolvimento de programas e, por isso, vem instalada no Arduino IDE. Ela fornece funções como d*igitalRead*, *digitalWrite*, *Serial.begin* e *analogRead*, entre outras. Já as bibliotecas padrão, também incluídas na instalação do Arduino IDE, não são incluídas por padrão em projetos novos que são criados, porque a memória do Arduino é limitada e, por isso, essas bibliotecas só são incluídas de forma explícita quando necessário. A inclusão de uma biblioteca padrão é feita por meio de uma declaração *#include* no início do código, seguido pelo nome da biblioteca com a extensão ".h" e envolvido entre os carateres '<' e '>'. Depois da inclusão da biblioteca no código, pode-se usar as funções que ela fornece no código. As bibliotecas padrão que o Arduino fornece são as seguintes:

- **EEPROM:** usada para ler e gravar dados na memória EEPROM do Arduino;
- ***Ethernet*:** permite ligar o Arduino à Internet ou à rede local, usando um *shield ethernet*;

- ***Firmata*:** permite a comunicação entre o Arduino e as aplicações de um computador via protocolo de comunicação série;
- **GSM:** permite ligar a uma rede GSM/GPRS, usando um *shield* GSM;
- ***LiquidCrystal*:** permite controlar displays de cristal líquido (LCD – *Liquid Crystal Display*);
- **SD:** permite ler e escrever dados em cartões de memória SD/SDHC;
- **Servo:** permite controlar os servomotores;
- **SPI:** permite a comunicação com dispositivos, usando o barramento SPI;
- ***SoftwareSerial*:** permite a comunicação série, usando os pinos digitais da placa;
- ***Stepper*:** permite controlar motores de passo;
- **TFT:** permite desenhar e escrever texto em um ecrã TFT;
- **Wi-Fi:** permite ligação à rede local e Internet por meio de um *shield* Wi-Fi;
- ***Wire*:** permite enviar e receber dados por meio de uma interface TWI/I2C em uma rede de dispositivos e sensores.

Por último, as bibliotecas de terceiros são criadas e disponibilizadas por muitos programadores que contribuem voluntariamente com *software* para a plataforma e não são disponibilizadas por defeito no Arduino IDE. Para usá-las é necessário fazer o *download* da biblioteca pretendida e fazer a instalação através do Arduino IDE. Elas oferecem funções adicionais às bibliotecas existentes ou novas funcionalidades que não estão presentes em nenhuma biblioteca padrão.

As bibliotecas usadas com os algoritmos das cifras de encriptação pertencem a esta última categoria. Para a cifra AES foi usada a biblioteca do *MarkT* [68]. Optou-se por esta biblioteca porque é muito utilizada para a encriptação AES e suporta o AES-128, AES-192 e o AES-256. Para o DES foi usada a biblioteca do *Octoate* [69]. Esta biblioteca também tem incluído

o algoritmo 3DES. Para o XTEA a biblioteca usada foi do *franksmicro* [70] e para o XXTEA a biblioteca usada foi do *alessandro1105* [71].

Para testar os algoritmos foram usados os exemplos que vêm com as bibliotecas, fazendo apenas algumas modificações e foi acrescentado a função *micros()* para calcular o tempo que demora a realizar a encriptação e a desencriptação.

3.4. A FUNÇÃO *MICROS()*

A função *micros()* retorna o número de microssegundos desde que a placa Arduino começou a correr o programa atual. Este número é reiniciado ao fim de cerca de 70 minutos, porque a função possui um tamanho de 32 bits. Nas placas de Arduino com 16 MHz, como é o caso do Arduino Mega 2560, esta tem uma resolução de 4 microssegundos, ou seja, o valor que é retornado é sempre um múltiplo de 4. Nas placas de Arduino com 8 MHz, esta função tem uma resolução de 8 microssegundos.

A função *micros()* foi usada durante a função de encriptação e de desencriptação dos algoritmos para calcular o tempo que demora a realizar a encriptação e a desencriptação dos algoritmos. A seguir é ilustrado um exemplo de implementação desta função, que neste caso é para calcular o tempo que demorou a realizar a encriptação para o algoritmo AES. Antes da execução da função de encriptação é registado o tempo que o programa já demorou para correr na placa do Arduino até aí. Depois de realizada a encriptação é outra vez registado o tempo que o programa está a correr e subtraído ao tempo anterior, resultando o tempo que demorou a realizar a encriptação.

```
long t0 = micros () ;
byte succ = aes.set_key (key, bits) ;
long t1 = micros()-t0 ;
```

3.5. AES

Usou-se a biblioteca do *MarkT* para fazer o teste ao algoritmo AES. Na figura 22 é ilustrado os resultados de saída para as entradas "*on*", "*password*" e "*0001:password:on*" e já com os tempos que demorou a realizar a encriptação e a desencriptação, respetivamente para cada entrada.

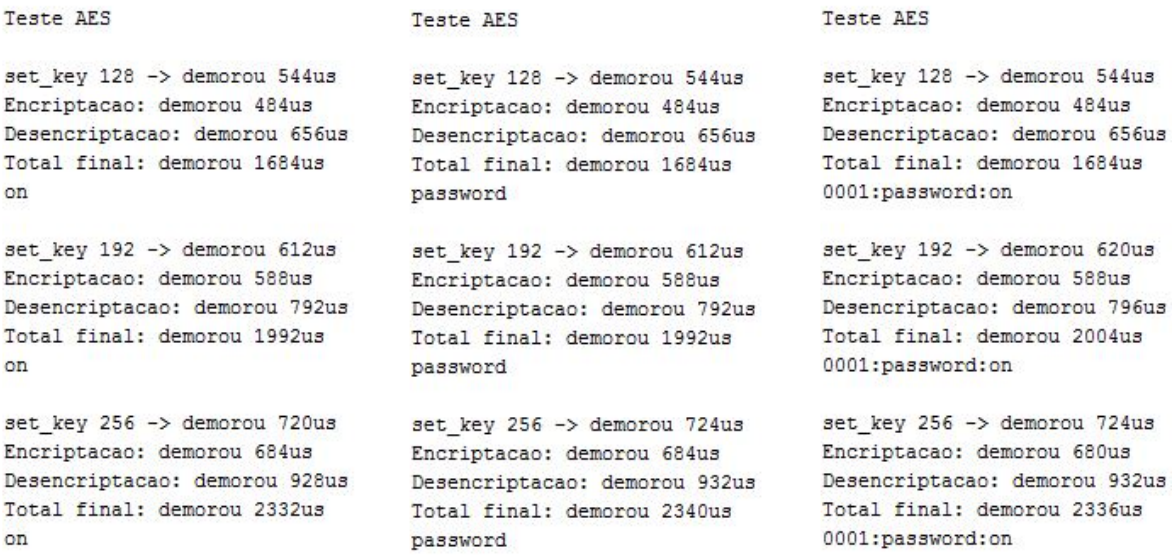


Figura 22 Resultados do AES

Na figura 22 pode ver-se e analisar algumas informações:

- O *set_key 128*, *set_key 192* ou *set_key 256* mostra o tempo que demorou a criar a chave de 128, 192 ou 256 bits respetivamente. Como já era de prever, quanto maior é a chave maior é o tempo que demora a criar a mesma;

- Para testar a eficiência da encriptação foi criado o campo “Encriptacao”, que vai conter o tempo que demorou a encriptar a palavra. Ao analisar este dado, a primeira conclusão que se tira é que quanto maior é a segurança da encriptação maior é o tempo que demora a encriptar, como já era de prever. Com uma análise mais criteriosa pode-se perceber que o tempo que demora a encriptação é mais ou menos linear à media que se aumenta o tamanho da chave, enquanto o tempo que demora para criar a chave sobe exponencialmente. Por exemplo, com o aumento da chave em 50% (128 bits para 192 bits), o tempo que demorou para criar a chave foi 12,5% a mais do que a chave de 128 bits (544µs para 612µs respetivamente) e o tempo que demorou a encriptação foi cerca de 20,66% a mais do que a chave de 128 bits (484µs para 588µs respetivamente). Já quando se aumenta a chave em 100% (128 bits para 256 bits) os valores são 33,09% para o tempo que demorou a mais para criar a chave e 41,32% para o tempo que demorou a mais a encriptar. Logo, percebe-se que o tempo para criar a chave quase que triplicou, enquanto que o tempo de encriptação só dobrou.

- Olhando para os resultados do teste percebe-se que o tamanho da palavra de entrada não interfere com o tempo que demora a fazer a encriptação. Isto acontece, porque o teste só utiliza um bloco de entrada para os 3 testes. Como a encriptação é feita em blocos percebe-se o porquê de demorar exatamente o mesmo tempo. Devido a isso, pode-se deduzir que a encriptação é mais eficiente à medida que a palavra de entrada é maior até ao máximo de 128 bits, que é o tamanho do bloco usado pela biblioteca *MarkT*. Testou-se a encriptação com 2 blocos e o resultado do tempo de encriptação é um bocadinho mais do dobro. Para o caso da entrada "*0001:password:on*" deu os seguintes valores:
    - set_key 128 -> 544µs
        - Encriptacao: 1048µs
    - set_key 192 -> 620µs
        - Encriptacao: 1248µs
    - set_key 256 -> 724µs
        - Encriptacao: 1432µs
- Para testar a eficiência da desencriptação foi criado o campo "Desencriptacao", que regista o tempo que demorou a fazer a desencriptação. Como acontece com a encriptação, também na desencriptação o tamanho da palavra de entrada não interfere com o tempo que demora a fazer a desencriptação. A principal conclusão que se tira ao olhar para os valores registados é que a desencriptação demorou cerca de 35% a mais do que a encriptação.

3.6. DES E TDES

Foi usada a biblioteca do *Octoate* para testar o algoritmo DES. Esta biblioteca também tem incluído o algoritmo TDES. Por isso, este teste foi feito ao mesmo tempo que o DES normal. O teste ao DES e ao TDES foi um pouco limitado devido a ele só trabalhar com blocos de 64 bits. Só foi possível testar a palavra de entrada "*password*", pois esta palavra tem 8 letras, logo 8 bytes, ou seja, 64 bits. Na figura 23 mostra o resultado de saída obtido.

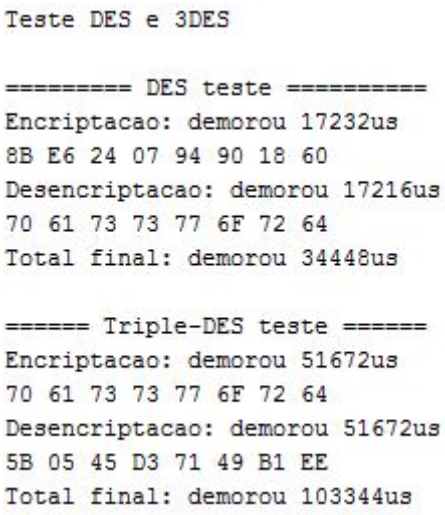


Figura 23 Resultados do DES e do TDES com a palavra de entrada "*password*" em hexadecimal

Neste teste pode-se analisar que os tempos de encriptação e desencriptação estão praticamente empatados no DES e então no TDES são iguais. Já a diferença de tempos do DES para o TDES é aproximadamente o triplo, o que se compreende tendo em conta que o TDES usa 3 chaves de 64 bits, enquanto que o DES normal só utiliza 1 chave.

3.7. XTEA

A biblioteca do *franksmicro* foi a biblioteca usada para testar o XTEA. Esta biblioteca utiliza 32 rondas em vez das 64 rondas recomendadas e utiliza blocos de entrada de 8 bytes, com um máximo de 2 blocos de cada vez, isto fez com que o teste padrão das palavras de entrada feito neste relatório não fosse igual para este algoritmo. Assim as palavras de entrada foram: "*00on*" (1 bloco – 8 bytes) e "*password*" (dividida em 2 blocos – 16 bytes). Na figura 24 tem-se os resultados obtidos com este teste.

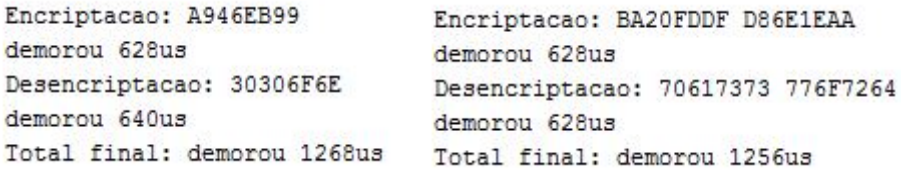


Figura 24 Resultados do XTEA com as entradas "*00on*" (lado esquerdo) e "*password*" (lado direito)

Ao olhar para os resultados obtidos nota-se que os tempos de encriptação e desencriptação foram iguais, exceto na desencriptação da palavra "*00on*". Não se percebe bem o porquê de isto ter acontecido nem o porquê de os dois blocos da palavra "*password*" demorar exatamente o mesmo tempo da palavra mais pequena com um bloco.

3.8. XXTEA

A biblioteca utilizada para testar o algoritmo XXTEA foi a do *alessandro1105*. Esta biblioteca suporta chaves até 16 carateres e não suporta as palavras "*on"* e "*0001:password:on*". Não suporta a palavra "*on*" por ter só 2 carateres e não suporta a palavra "*0001:password:on*" devido aos ':'. Então as palavras utilizadas para fazer os testes de velocidade foram: "*password*" e "*0001passwordon*" com a chave "chave". Também foi feito um teste com a chave "*password*" para a palavra de entrada "*0001passwordon*", para analisar se havia diferenças significativas de tempo de encriptação e desencriptação ao usar uma chave maior. Na figura 25 são mostrados os resultados obtidos destes testes.

```
Chave: chave
Palavra de entrada: password
Encriptacao: W©_e÷^ê`êèÖ$
demorou 2592us
Desencriptacao: password
demorou 2772us
Total final: demorou 5364us
```

```
Chave: chave
Palavra de entrada: 0001passwordon
Encriptacao: ¸öàìßÂM{□□▶qpL¿□«`ÓU¹
demorou 3636us
Desencriptacao: 0001passwordon
demorou 3844us
Total final: demorou 7480us
```

```
Chave: password
Palavra de entrada: 0001passwordon
Encriptacao: v]êK›,Kl◦ÑkñL‹ÂÉß/œ¸
demorou 3628us
Desencriptacao: 0001passwordon
demorou 3848us
Total final: demorou 7476us
```

Figura 25 Resultados do XXTEA

Analisando os resultados anteriores conclui-se que a desencriptação dura mais que a encriptação (cerca de 6,94% para a palavra "*password*" e 5,72% para a palavra "*0001passwordon*") e que o aumento da chave não interfere com os tempos que demora a encriptar ou a desencriptar. Contudo, aumentou o tempo de encriptação, cerca de 40,28%, quando foi aumentada a palavra de entrada em 75%.

3.9. COMPARAÇÃO DIRETA DOS ALGORITMOS TESTADOS

Foram feitos 3 gráficos (um para cada uma das palavras testadas) para comparar diretamente os tempos das encriptações e desencriptações dos diferentes algoritmos testados. O gráfico "*on/0on*" tem as palavras de entrada "*on*" e "*00on*", devido ao algoritmo XTEA só poder ter sido testado com a palavra "*00on*", porque este só utiliza blocos de entrada de 8 bytes. Acontece o mesmo para o gráfico "*0001:password:on/0001passwordon*", este tem as palavras "*0001:password:on*" e "*0001passwordon*", pois com o XXTEA não foi possível testar a palavra "*0001:password:on*", devido a um erro que aparecia quando se metia ":". Nas figuras 26, 27 e 28 são apresentados estes 3 gráficos.

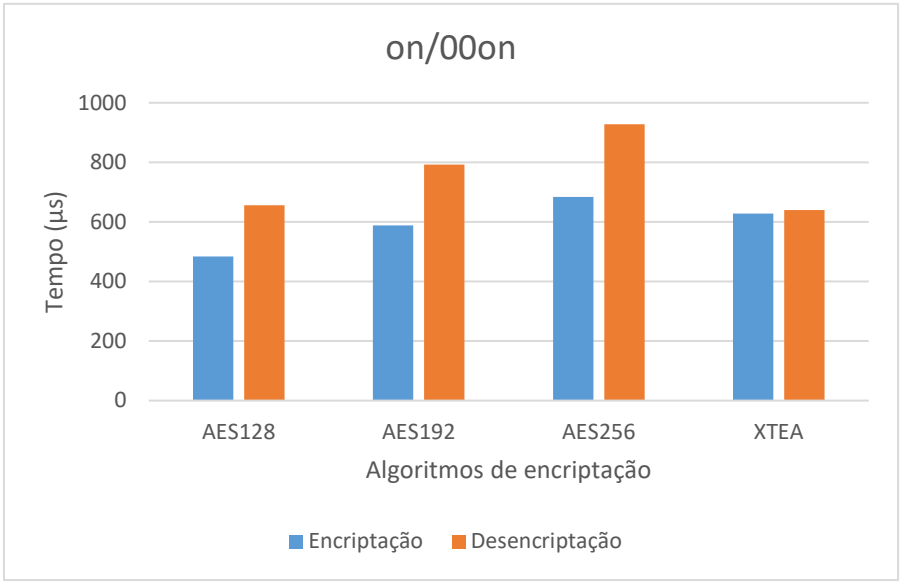


Figura 26 Tempos obtidos para a palavra de entrada "*On/00on*"

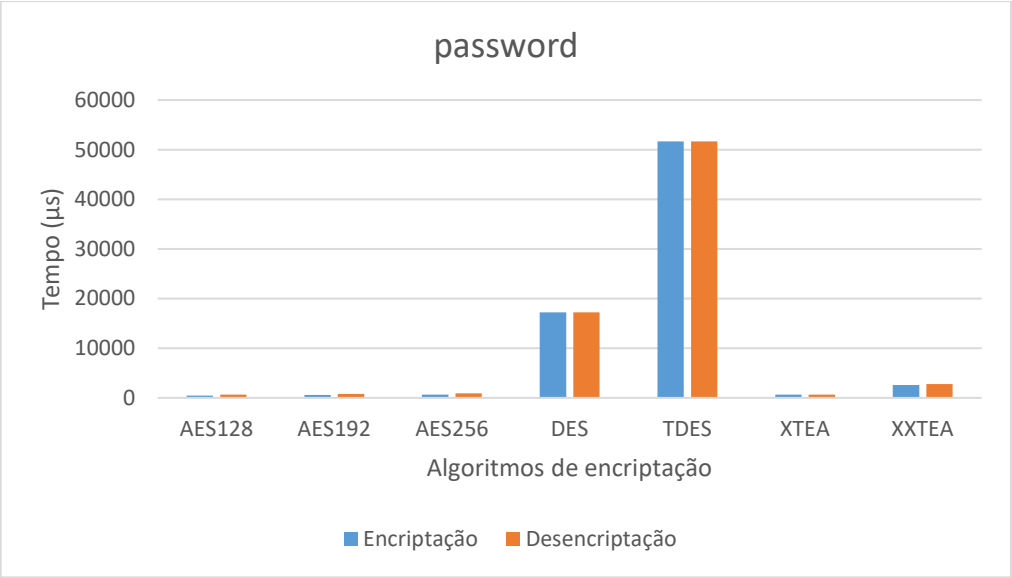


Figura 27 Tempos obtidos para a palavra de entrada "*password*"

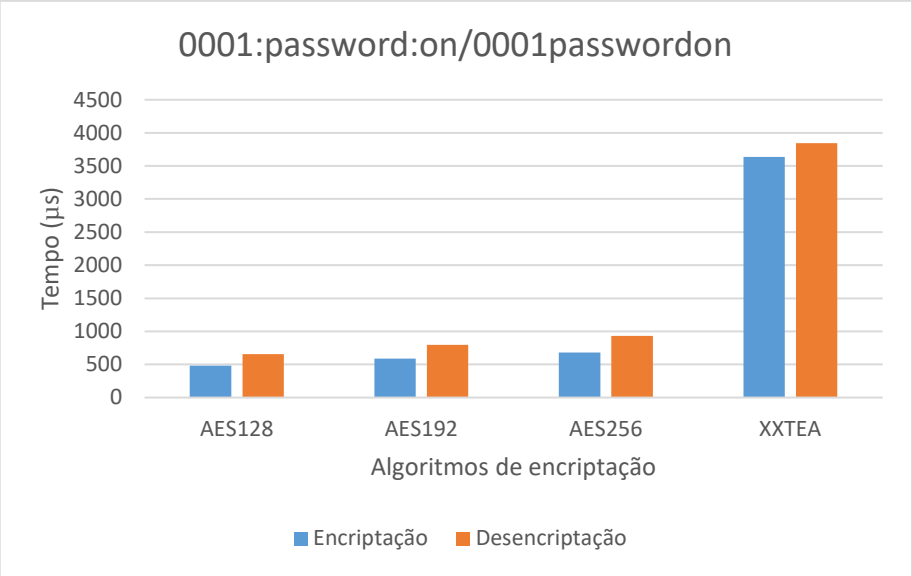


Figura 28 Tempos obtidos para a palavra de entrada "*0001:password:on/0001passwordon*"

Começando por analisar o gráfico da palavra "*on/00on*", pode perceber-se que, mesmo usando uma palavra de entrada com o dobro do tamanho, o algoritmo XTEA consegue rivalizar a nível de tempo com o AES e até ganha no tempo de desencriptação a todas as versões do AES. Contudo, o AES oferece mais segurança do que o XTEA, por isso, compensa usar o AES em detrimento do XTEA, visto que o desempenho não é assim tão díspar.

Ao analisar o gráfico da palavra "*password*", o que se nota logo é a lentidão do DES e do TDES. O TDES chega a demorar quase 107 vezes mais do que o AES de 128 bits para realizar a encriptação. Em relação ao XTEA, continua a ser bastante rápido, tendo obtido tempos muito idênticos (ou igual para o caso do tempo da encriptação) ao gráfico da palavra "*on*/*00on*", conseguindo bater as versões todas do AES. O AES só ficou um pouco à frente nos tempos de encriptações das versões de 128 e 192 bits. Por fim, o XXTEA obteve tempos de cerca de 4 vezes mais do que o XTEA. Neste teste, o AES volta a ganhar no global a todos os adversários, mesmo que os tempos do XTEA rivalize com o AES, este último oferece mais segurança.

No gráfico correspondente à palavra "*0001:password:on*/*0001passwordon*" é possível comparar diretamente o AES com o XXTEA, apesar deste último usar uma palavra de entrada menor. O XXTEA é bastante mais lento do que o AES, demorando cerca de 8 vezes mais do que o AES de 128 bits para realizar a encriptação e cerca de 6 vezes para realizar a desencriptação. Mais uma vez, o AES volta a ganhar no teste comparativo.

Nestes comparativos, o algoritmo AES ganhou em todos os testes. Ele é, sem dúvida, o que oferece a melhor relação segurança/desempenho, pois tem um excelente desempenho em conjugação com uma excelente segurança. Também dá para perceber claramente nestes testes que o DES e o TDES demoram bastante tempo a encriptar e a desencriptar, demonstrando que não são uma opção viável para alguns cenários de utilização. Por fim, o XXTEA tem uma segurança muito boa e o desempenho é claramente melhor do que o TDES, mostrando-se um bom substituto ao AES.

3.10. APLICAÇÕES PARA OS ALGORITMOS

O AES é tipicamente utilizado quando a segurança e o desempenho são muito importantes. O AES é utilizado nas redes Wireless, pois permite uma segurança boa, importante para a rede não ficar comprometida, sem diminuir a velocidade dela. O AES também é utilizado em teclados Wireless com encriptação pois, para além de permitir uma boa segurança, permite um *delay* baixo devido à sua velocidade rápida de encriptação e desencriptação. As mesmas caraterísticas permitem que seja muito utilizado na encriptação dos discos rígidos.

O XXTEA é bastante utilizado para proteção de emails, chats, etc. apesar de agora já estar a ser substituído pelo AES devido às vantagens já faladas anteriormente. É também muito utilizado em dispositivos que não suportam o *Hyper Text Transfer Protocol Secure* (HTTPS), como o Arduino, pois o XXTEA é fácil de implementar e tem uma boa segurança.

O DES e TDES são pouco utilizados, por serem muito lentos e fracos a nível de segurança em relação ao AES. O TDES chegou a ser utilizado no IPSsec, tendo sido substituído pelo AES-CBC de 128 bits.

4. PROTOCOLOS DE COMUNICAÇÃO SEM FIOS DO ARDUINO

O Arduino sendo uma plataforma de programação muito simples e ao mesmo tempo cheia de recursos e acessórios, é muito interessante para ser usada na Internet das Coisas. Pode, por exemplo, ser utilizada para enviar informações para a nuvem ou para um servidor para que possam ser analisadas e interpretadas de uma maneira inteligente (verificar as nossas tendências, as nossas preferências, etc.). Pode também conceber-se aplicações em que é analisado o canal favorito de televisão do utilizador, qual a temperatura mais confortável, a que horas o utilizador chega a casa, se já se foi deitar e trancar automaticamente as portas, ligar o alarme, desligar as luzes e analisar os consumos energéticos. Isto tudo já é possível com o Arduino e todos os dias surgem novos projetos a implementar, pouco a pouco, estes sistemas de domótica, em que os Arduinos comunicam entre si e a Internet.

Devido a esta crescente utilização do Arduino para várias situações do quotidiano, torna-se cada vez mais interessante não só proteger a mensagem enviada através de alguma cifra de encriptação, mas também conhecer e utilizar os melhores tipos de protocolos de comunicação sem fios do Arduino para cada situação específica, de modo a permitir a comunicação entre Arduinos, atuadores e sensores sem a necessidade de cabos de rede. A seguir serão apresentados os três tipos principais de rede sem fios existentes para o Arduino e comparados uns com os outros para tentar saber quando se deve usar um ou outro.

4.1. WI-FI

O Wi-Fi é um conjunto de especificações para redes locais sem fios, baseadas na norma IEEE 802.11. A *Wi-Fi Alliance* é a entidade responsável pelo licenciamento de produtos com esta tecnologia. Usando a tecnologia Wi-Fi é possível criar redes que ligam computadores e outros dispositivos compatíveis (smartphones, tablets, consolas de jogos, impressoras, arduinos, sensores, atuadores, etc.) que estejam próximos geograficamente. Estas redes não necessitam de cabos, já que este tipo de rede transmite dados por radiofrequência. Esta tecnologia permite que a rede seja utilizada em qualquer ponto dentro dos limites de alcance da transmissão, possibilitando a inserção rápida de novos dispositivos na rede e evitando que paredes ou estruturas prediais sejam furadas ou adaptadas para a passagem de cabos. A flexibilidade do Wi-Fi é tão grande que se tornou viável a implementação destas redes em muitos lugares. Atualmente é comum encontrar redes Wi-Fi em shoppings, hotéis, aeroportos, bares, restaurantes, escolas, universidades, escritórios, hospitais, etc. Para utilizar estas redes, basta o utilizador usar um dispositivo compatível com esta tecnologia.

Na tecnologia 802.11, a transmissão é feita por sinais de radiofrequência, que se propagam pelo ar e podem cobrir áreas de centenas de metros. Como existem muitos serviços que usam sinais de rádio, cada um deve operar de acordo com as exigências estabelecidas pelo governo de cada país para evitar interferências entre os diferentes serviços. Contudo, há alguns segmentos de frequência que podem ser usados sem a necessidade da aprovação direta das entidades competentes de cada governo. As faixas com licença gratuita (ISM – *Industrial, Scientific and Medical*) operam, entre outros, nos seguintes intervalos: 902 MHz - 928 MHz, 2.4 GHz – 2.485 GHz e 5.15 GHz – 5.825 GHz (dependendo do país, esses limites podem sofrer variações). As duas últimas faixas são as faixas que o Wi-Fi utiliza. A seguir serão apresentadas as versões mais usadas atualmente: 802.11b, 802.11g, 802.11n e 802.11ac.

4.1.1. 802.11B

Em 1999 foi lançada uma atualização da norma 802.11 original que recebeu o nome de 802.11b. Esta versão permite velocidades de comunicação de 1 Mb/s, 2 Mb/s, 5.5 Mb/s e 11 Mb/s. O intervalo de frequências é o mesmo utilizado pela versão 802.11 original, ou seja, entre 2.4 GHz e 2.4835 GHz. A técnica utilizada para transmissão desta tecnologia é a *Direct*

Sequence Spread Spectrum (DSSS). Esta técnica cria vários segmentos de dados para transmitir e enviar simultaneamente por todos os canais disponíveis.

A distância de transmissão que a norma 802.11b pode atingir é de, teoricamente, 400 metros em ambientes abertos e pode atingir 50 metros em lugares fechados, como escritórios ou residências. A taxa de transmissão de dados diminui até chegar ao seu limite mínimo de 1 Mb/s à medida que a estação fica mais longe do ponto de acesso, de modo a que a ligação não desligue.

4.1.2. 802.11G

A versão 802.11g foi lançada em 2003 para substituir o 802.11b e é totalmente compatível com este padrão. Isso significa que um dispositivo que opera com 802.11g pode comunicar com outro que suporta 802.11b sem nenhum problema, exceto o facto de que a taxa de transmissão de dados é limitada ao que é suportada pela versão 802.11b.

O 802.11g tem uma taxa de transmissão de até 54 Mb/s e trabalha com frequências na faixa de 2.4 GHz (canais com 20 MHz), tendo praticamente o mesmo alcance do seu sucessor. A técnica de transmissão é a *Orthogonal Frequency Division Multiplexing* (OFDM), todavia quando é feita a comunicação com um dispositivo 802.11b, a técnica de transmissão passa a ser a DSSS. No OFDM os dados a serem enviados são divididos em vários pequenos conjuntos de dados que são transmitidos, simultaneamente, em diferentes frequências.

4.1.3. 802.11N

A versão 802.11n começou a ser desenvolvida em 2004 e foi finalizada em Setembro de 2009. Usa o esquema *Multiple-Input Multiple-Output* (MIMO) para aumentar as taxas de transferência de dados, usando várias combinações de antenas. Esta caraterística faz com que consiga transmitir a 300 Mb/s e, teoricamente, pode conseguir transmitir até 600 Mb/s. Usando o modo mais simples, é possível chegar aos 150 Mb/s.

O 802.11n utiliza as frequências 2.4 GHz e 5 GHz, o que o faz compatível com as versões anteriores. Cada canal usa 40 MHz de largura. A sua técnica de transmissão é o OFDM, mas com determinadas alterações, devido à utilização do MIMO, sendo por isso muitas vezes chamado de MIMO-OFDM. Já a área de cobertura do 802.11n pode passar dos 400 metros.

4.1.4. 802.11AC

A norma 802.11ac foi criada para substituir o 802.11n tendo sido desenvolvida entre 2011 e 2013. Este padrão possibilitou um grande aumento de velocidade, estimada em 433 Mb/s no modo mais simples. Contudo, é possível, em teoria, fazer a rede superar os 6 Gb/s usando várias antenas, no máximo 8. O 802.11ac, tal como o 802.11n, também pode trabalhar na frequência de 5 GHz com um tamanho de canal padrão de 80 MHz, podendo ser utilizado um tamanho de 160 MHz como opcional.

O 802.11ac utiliza técnicas mais avançadas de modulação do que o seu antecessor. Ele trabalha com o esquema *Multi-User MIMO* (MU-MUMO) que permite a transmissão e a receção do sinal de vários terminais, como se trabalhassem de uma forma colaborativa e na mesma frequência. Também este padrão usa um método de transmissão conhecido como *Beamforming* (também conhecido como TxBF), que no padrão 802.11n é opcional. Esta tecnologia permite ao aparelho transmissor avaliar a comunicação com um dispositivo cliente para otimizar a transmissão na sua direção.

4.1.5. WI-FI NO ARDUINO: MÓDULO WI-FI ESP8266

O módulo Wi-Fi ESP8266 (figura 29) para o Arduino é um módulo transcetor Wi-Fi, baseado no *System-on-a-Chip* (SoC) ESP8266. Este chip implementa a pilha de protocolos TCP/IP completa e tem um grande poder computacional. Devido a estas caraterísticas, é possível usar esta placa como uma placa de ligação Wi-Fi simples, não sendo necessário usar o processador principal para controlar a gestão da comunicação Wi-Fi. Assim, o utilizador pode gerir sensores e elaborar de forma autónoma os seus sinais e medições. Este módulo permite construir soluções para a Internet das Coisas muito compactas com suporte para as redes Wi-Fi 802.11 b/g/n.

Figura 29 MÓDULO WI-FI ESP8266 [62]

4.2. *BLUETOOTH LOW ENERGY* (BLE)

O *Bluetooth Low Energy* (BLE) é uma tecnologia de comunicação sem fios que foi lançada em 2010 junto com a versão 4.0 do Bluetooth. O BLE permite reduzir os consumos em dispositivos que não precisam de transmitir grandes volumes de dados, conseguindo alcançar um gasto energético de apenas 10% em relação ao Bluetooth clássico, possibilitando assim que esses dispositivos sejam alimentados com baterias do tamanho de moedas. O BLE consegue esta economia de energia colocando o dispositivo em modo *sleep* durante grande parte do tempo, saindo desse modo para fazer as ligações que duram apenas milissegundos. O BLE foi projetado para aplicações que precisam de enviar poucas informações e esporadicamente. Isso pode ser confirmado analisando os consumos energéticos do BLE que tem um consumo energético com picos de 15 mA, mas com uma média de apenas 1 µA.

Um controlador BLE tem caraterísticas do Bluetooth clássico, mas não é e nem atua como o Bluetooth clássico não sendo compatível com ele. Como não há compatibilidade entre as duas tecnologias, se um determinado projeto quer ter implementado no seu dispositivo ambas arquiteturas isto é possível através da execução de um protocolo que faz a mediação entre o Bluetooth clássico e o BLE. Os dispositivos com este suporte são denominados por *dual-mode* ou *smart ready*. Se o dispositivo só implementar o suporte ao BLE, este é denominado por *single-mode* ou *smart*. A diferença entre estas duas implementações é que a primeira não costuma ter os mesmos ganhos energéticos do que a segunda. Na tabela 30 apresenta as diferenças a nível de especificação técnica entre o Bluetooth clássico e o BLE.

Tabela 30 Diferenças entre BT e BLE

Especificação técnica	Bluetooth clássico	Bluetooth Low Energy
Frequência	2400 a 2483,5 MHz	2400 a 2483,5 MHz
Técnica de modução	Salto de frequência	Salto de frequência
Esquema de modulação	GFSK	GFSK
Índice de modulação	0,35	0,5
Número de canais	79	40
Largura de banda do canal	1 MHz	2 MHz
Taxa de dados nominal	1-3 Mb/s	1 Mb/s
Rendimento da aplicação	0,7 – 2,1 Mb/s	< 0,3 Mb/s
Nós / escravos ativos	7	Ilimitado
Segurança	56 a 128 bits	AES - 128 bits
Robustez	FHSS	FHSS
Voz	Capaz	Incapaz

4.2.1. BLE NO ARDUINO: MÓDULO BLE BLUETOOTH V4.0 HM-11

O módulo Bluetooth V4.0 HM-11 (figura 30) para o Arduino é um dispositivo de montagem superficial (SMD – *Surface-Mount Device*), baseado no chip "TI cc2541", barato, pequeno e fácil de usar. O *firmware*, que vem pré-carregado, permite que o utilizador construa rapidamente comunicações BLE através do seu comando AT. O módulo suporta comunicações BLE com iPhones e iPads com iOS 5.0 ou superior e Android 4.3 ou superior.

Figura 30 Módulo BLE Bluetooth V4.0 HM-11 [62]

4.3. ZIGBEE

A rede ZigBee foi criada pelo *Institute of Electrical and Electronics Engineers* (IEEE) e pela *ZigBee Alliance* com o objetivo de disponibilizar uma rede com muita baixa potência de operação, possibilitando um baixo consumo de energia nos dispositivos, estendendo a vida útil das suas baterias, que podem mesmo durar anos. Esta rede foi criada tendo em vista os dispositivos que não precisam de taxas de transmissão de dados tão altas quanto às permitidas pelo Bluetooth e precisam de ter um baixo consumo energético. Este protocolo foi criado para ser utilizado na automação e nos controlos remotos. As principais caraterísticas do ZigBee são:

- Diferentes frequências de operação e taxas de dados (868 MHz a 20 Kb/s, 915 MHz a 40 Kb/s, 2.4 GHz a 250 Kb/s);
- Um mesmo nó pode ter vários papéis na mesma rede;
- Suporte das diferentes topologias de rede;
- Habilidade de auto-organizar e auto-reestruturar;
- Permite um número elevados de dispositivos ligados à rede (máximo de 65535 dispositivos por cada dispositivo coordenador);
- Alta durabilidade da bateria dos dispositivos;
- Interoperabilidade (capacidade de se comunicar de forma transparente com outros sistemas).

O ZigBee usa dois tipos diferentes de endereçamento: o endereçamento EUI-64, com 64 bits de endereços, idêntico à rede Ethernet, existindo também a possibilidade de se utilizar o endereço reduzido com comprimento de 16 bits. Este último é usado quando a rede está configurada e permite um total de 64 mil nós numa rede, aproximadamente.

4.3.1. ZIGBEE NO ARDUINO: MICROCHIP MRF24J40MA

O microchip MRF24J40MA (figura 31) é um dos mais utilizados e o que tem o melhor desempenho. O módulo tem a certificação 2,4 GHz IEEE 802.15.4. O módulo é compatível com centenas de microcontroladores *Peripheral Interface Controller* (PIC) através de uma interface SPI de 4 fios e é a solução ideal para redes de sensores sem fios, domótica, automação de edifícios e aplicações para consumidores.

Figura 31 Microchip MRF24J40MA [64]

4.4. WI-FI VS BLE VS ZIGBEE

Na tabela 31 pode ver-se uma comparação das várias caraterísticas das tecnologias Wi-Fi, BLE e ZigBee. Nesta tabela também estão inseridas as aplicações e os mercados para onde estas tecnologias foram destinadas quando foram criadas e onde são usadas atualmente.

Tabela 31 Comparação entre as várias tecnologias sem fios estudadas

	Wi-Fi (802.11n)	**BLE**	**ZigBee**
Norma	IEEE 802.11	IEEE 802.15.1	IEEE 802.15.4
Organização	Wi-Fi Alliance	Bluetooth SIG	ZigBee Alliance
Topologias	Árvore	Árvore	Estrela, malha e árvore
Frequências RF	2,4 GHz 5.8 GHz	2,4 GHz	868 MHz, 915 MHz 2,4 GHz
Máximo de taxa de dados	600 Mb/s	1 Mb/s	250 Kb/s
Máximo de alcance ao ar livre	100 m	50 m	500 m
Consumo médio de energia	Tx: 220+ mA Rx: 215+ mA	Tx: 15-20 mA Rx: 15-20 mA	Tx: 25-35 mA Rx: 20-30 mA
Vida da bateria	Horas	Dias - semanas	Meses - anos
Custo da certificação	Alto	Alto	Médio
Adoção da comunidade de desenvolvimento	Alto	Médio	Alto
Interoperabilidade	Alto	Médio	Alto
Confiabilidade	Médio	Médio	Baixo
Focos aplicacionais	Internet, email, vídeo, etc.	Substituição do cabo	Monitoramento e controlo
Mercados	Residencial e comercial	Saúde e fitness	Automação, comercial e industrial

4.5. SEGURANÇA USADA NAS DIFERENTES COMUNICAÇÕES SEM FIOS

Tanto no *Wi-Fi Protected Access 2* (WPA2) / 802.11i (a mais recente norma do Wi-Fi), ZigBee ou BLE é utilizado o algoritmo AES com um comprimento da chave de 128 bits. As três redes sem fios utilizam este algoritmo com o modo CCM. O AES-CCM é um algoritmo com criptografia e autenticação, criado para fornecer confidencialidade e integridade dos dados. O CCM junta dois modos de cifra de blocos: o modo contador para a confidencialidade dos dados e o modo *Cipher Block Chaining Message Authentication Code* (CBC-MAC) para a integridade dos dados.

O modo contador faz com que a cifra de bloco AES se comporte como uma cifra sequencial, combinando assim a segurança de uma cifra de bloco com a facilidade de uso de uma cifra sequencial. Este modo requer um contador. O contador começa num valor arbitrário, mas predeterminado, e é incrementado de uma forma específica. Por exemplo, um contador simples iniciaria com um valor 1 e aumentaria sequencialmente 1 valor para cada bloco. No entanto, a maioria das implementações obtém o valor inicial do contador a partir de um valor *nonce* (número exclusivo, ou seja, é um número que é usado só uma vez) que muda para cada mensagem sucessiva. A cifra AES é usada para encriptar o contador e produzir um fluxo de chaves. Quando a mensagem original chega, ela é dividida em blocos de 128 bits e, para cada bloco, é realizada uma operação XOR com o fluxo de chave correspondente, produzindo assim o texto cifrado. Na figura 32 estão representados os processos de encriptação e desencriptação, usando este modo contador.

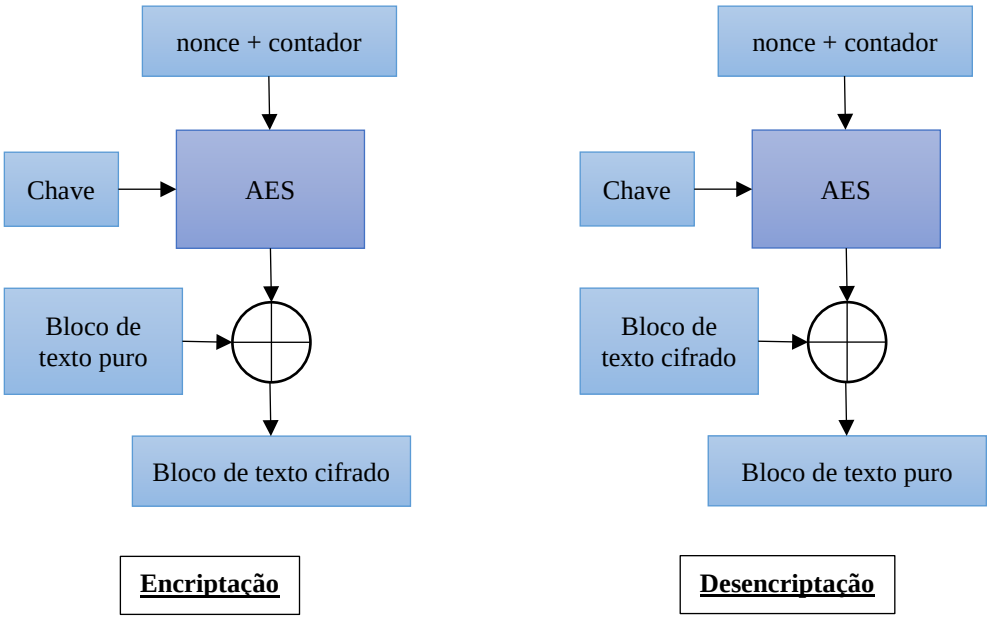


Figura 32 Encriptação e desencriptação do modo contador

A segurança do sistema está no contador. Enquanto o valor do contador nunca for repetido com a mesma chave, o sistema é seguro. Em resumo, as principais caraterísticas do modo contador são as seguintes:

- Permite que uma cifra de bloco trabalhe como uma cifra sequencial;
- O uso do contador permite que o fluxo de chaves seja gerado independente da mensagem, possibilitando assim que o fluxo de chaves seja gerado antes da mensagem chegar;
- Como o protocolo por si só não cria qualquer interdependência de encriptação entre os vários blocos de uma mensagem, os vários blocos da mensagem podem ser encriptados em paralelo se o hardware o permitir;
- Como o processo de desencriptação é exatamente o mesmo do que a encriptação, cada dispositivo só precisa de implementar o bloco de encriptação do AES;
- Como o modo contador não exige que a mensagem seja dividida num número exato de blocos, o comprimento do texto encriptado pode ser exatamente o mesmo que o comprimento da mensagem do texto puro.

O CBC-MAC é uma técnica utilizada para verificar a integridade da mensagem. Ela está representada na figura 33, onde as caixas azuis representam o protocolo de encriptação, que neste caso é o AES. Como se pode ver na figura, antes da encriptação é realizado uma operação XOR de um bloco de texto puro com o bloco resultante anterior, fazendo com que cada bloco dependa da construção do bloco anterior. Isto garante que qualquer alteração realizada no texto puro original causará uma modificação no último bloco de texto cifrado de maneira imprevisível para quem não conhece a chave utilizada. No primeiro bloco de texto puro, como não há ainda um bloco de texto cifrado anterior, é utilizado um vetor de iniciação. O vetor de iniciação é tipo um bloco falso que dá uma aleatoriedade ao processo, fazendo com que o mesmo texto que seja encriptado várias vezes dê origem a um código encriptado resultante diferente.

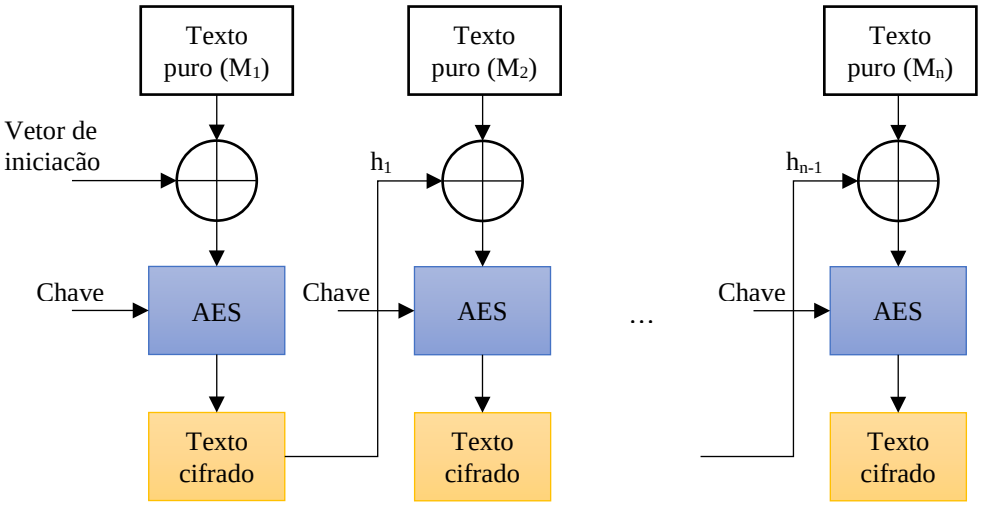


Figura 33 AES CBC-MAC

5. CONCLUSÕES

Dos algoritmos criptográficos analisados percebe-se que o AES é mais seguro do que os restantes, sendo também mais rápido. Este algoritmo, por ser muito conhecido e usado, é também o mais estudado no campo da criptoanálise não tendo, mesmo assim, sido ainda quebrado com sucesso, comprovando a sua elevada segurança. Pelo contrário, não existem muitos estudos relacionados com o XXTEA. Contudo, este algoritmo, que veio para corrigir problemas e melhorar a segurança do TEA e do XTEA, é muito seguro e tem um bom desempenho. Já o DES não oferece tanta segurança como o AES e o XXTEA, porém o TDES tem uma proteção muito segura e equivalente ao AES e ao XXTEA, porém muito lento.

Os testes de desempenho efetuados no Arduino permitem concluir que o AES é claramente mais rápido do que os outros algoritmos, pelo menos contra os que oferecem uma segurança teoricamente muito equivalente a este. O XXTEA mostrou-se um bom substituo ao AES, pois teve um bom desempenho nos testes, ficando logo atrás do AES, conseguindo oferecer também uma boa segurança. Já o TDES comprovou-se que é muito lento para os padrões atuais e já não compensa utilizá-lo principalmente nas aplicações onde o desempenho é essencial. Por isso, o TDES tem sido substituído gradualmente, ao longo dos últimos anos, pelo AES. O XXTEA continua a ser muito utilizado em dispositivos que não suportam o HTTPS, como o Arduino.

Depois destes testes foram estudadas três redes sem fios do Arduino: Wi-Fi, BLE e ZigBee, e identificados três módulos para o Arduino com suporte a estas redes sem fios. O Wi-Fi é uma tecnologia criada para ser utilizada em ambientes onde é necessária muita largura de banda, o BLE para ser utilizado em dispositivos onde a poupança energética é primordial e o ZigBee para ser utilizado na automação e controlos remotos. Todas as redes sem fios estudadas utilizam o algoritmo AES com uma chave de 128 bits para proteger os dados transmitidos. Para além disto, utilizam um contador para permitir enviar dados de uma forma sequencial e utilizam a técnica CBC-MAC para garantir a integridade dos dados que são enviados, ou seja, para garantir que não são modificados durante a transmissão.

Para um trabalho futuro seria interessante analisar a diferença de tempo de envio e taxa de transferência dos algoritmos de encriptação aqui estudados para diferentes redes sem fios, de modo a analisar qual ou quais as encriptações que mais congestionam uma rede e qual ou quais as mais rápidas a enviar a informação completa. Esta informação permitiria analisar qual a configuração mais indicada para cada sistema ou cenário de utilização.

Referências Documentais

[1] Bruno Lucattelli – *Writing the Enigma Machine in ABAP: An Educational Project*, 2016.

[2] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer e Manfred Schimmler – *Breaking Ciphers with COPACOBANA – A Cost-Optimized Parallel Code Breaker*. Cryptographic Hardware and Embedded Systems - CHES 2006. 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings. Springer Berlin Heidelberg. pp 101-118. 2006. ISBN: 978-3-540-46559-1, 978-3-540-46561-4.

[3] S. Kelly - *Security Implications of Using the Data Encryption Standard (DES),* Aruba Networks, 12-2006.

[4] Atul Kahate – *Cryptography and Network Security*. Tata McGraw-Hill Education. 2013. ISBN: 1259029883, 9781259029882.

[5] Eli Biham, Adi Shamir - *Differential Cryptanalysis of the Data Encryption Standard*. Springer New York. 1993. ISBN: 978-1-4613-9316-0, 978-1-4613-9314-6.

[6] Mitsuru Matsui, Atsuhiro Yamagishi - *A New Method for Known Plaintext Attack of FEAL Cipher*. Advances in Cryptology — EUROCRYPT' 92. Workshop on the Theory and Application of Cryptographic Techniques Balatonfüred, Hungary, May 24–28, 1992 Proceedings. Springer Berlin Heidelberg. 1993. pp 81-91. ISBN: 978-3-540-56413-3, 978-3-540-47555-2.

[7] Akihiro Shimizu, Shoji Miyaguchi - *Fast Data Encipherment Algorithm FEAL*. Advances in Cryptology — EUROCRYPT' 87. Workshop on the Theory and Application of Cryptographic Techniques Amsterdam, The Netherlands, April 13–15, 1987 Proceedings. Springer Berlin Heidelberg. 1988. Section VI:. pp 267-278. ISBN: 978-3-540-19102-5, 978-3-540-39118-0.

[8] National Institute of Standards and Technology (NIST) – *FIPS Publication 46-2: Data Encryption Standard*. 30-12-1993. Emitido originalmente pelo National Bureau of Standards.

[9] Mitsuru Matsui – *Linear Cryptanalysis Method for DES Cipher*. Advances in Cryptology — EUROCRYPT ’93. Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings. Springer Berlin Heidelberg. 1994. pp 386-397. ISBN: 978-3-540-57600-6, 978-3-540-48285-7.

[10] E. Okamoto – *Personal communication*. 03-1994.

[11] M. Matsui – *Linear cryptanalysis of DES cipher (I)*. 01-1994.

[12] A.W. Drake – *Fundamentals of Applied Probability Theory*. McGraw-Hill Book Company, New York. 1967.

[13] Stephane Moore – *Meet-in-the-Middle Attacks,* 16-09-2010.

[14] Eli Biham, Alex Biryukov – *An Impovement of Davies’ Attack on DES,* 1994.

[15] United States National Institute of Standards and Technology (NIST) – *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. Federal Information Processing Standards Publication 197. 26-09-2001.

[16] Joan Daemen e Vincent Rijmen - *Security of a Wide Trail Design*. Progress in Cryptology — INDOCRYPT 2002. Third International Conference on Cryptology in India Hyderabad, India, December 16–18, 2002 Proceedings. Springer Berlin Heidelberg. 2002. pp 1-11. ISBN: 978-3-540-00263-5, 978-3-540-36231-9.

[17] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen e Mahmoud Modarres-Hashemi - *Improved Impossible Differential Cryptanalysis of 7-Round AES-128*. Progress in Cryptology - INDOCRYPT 2010. 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings. Springer Berlin Heidelberg. 2010. pp 282-291. ISBN: 978-3-642-17400-1, 978-3-642-17401-8.

[18] Zheng Yuan – *New Impossible Differential Attacks on AES*. Beijing Electronic Science and Technology Institute, Beijing 100070, China. Center for Advanced Study Tsinghua University, Beijing, 100084, China. 01-2010.

[19] B. Bahrak e Mohammad Reza Aref – *Impossible differential attack on seven-round AES-128*. Artigo em IET Information Security 2(2): 28-32. 07/2008.

[20] Wentao Zhang, Whenling Wu e Dengguo Feng – *New Results on Impossible Differential Cryptanalysis of Reduced AES*. Information Security and Cryptology - ICISC 2007. 10th International Conference, Seoul, Korea, November 29-30, 2007. Proceedings. Springer Berlin Heidelberg. 2007. pp 239-250. ISBN: 978-3-540-76787-9, 978-3-540-76788-6.

[21] Jiqiang Lu, Orr Dunkelman, Nathan Keller e Jongsung Kim – *New Impossible Differential Attacks on AES*. Progress in Cryptology - INDOCRYPT 2008. 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings. Springer Berlin Heidelberg. 2008. pp 279-293. ISBN: 978-3-540-89753-8, 978-3-540-89754-5.

[22] Raphael C.-W. Phan – *Impossible differential cryptanalysis of 7-round advanced encryption standard (AES).* Jornal Information Processing Letters. Elsevier North-Holland, Inc. Amsterdam, The Netherlands, The Netherlands. Volume 91 Issue 1, 16/07/2004. pp 33 – 38.

[23] Alex Biryukov – *The Boomerang Attack on 5 and 6-Round Reduced AES*. Advanced Encryption Standard – AES. 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers. Springer Berlin Heidelberg. 2005. pp 11-15. ISBN: 978-3-540-26557-3, 978-3-540-31840-8.

[24] Michael Gorski e Stefan Lucks – *New Related-Key Boomerang Attacks on AES*. Progress in Cryptology - INDOCRYPT 2008. 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings. Springer Berlin Heidelberg. 2008. pp 266-278. ISBN: 978-3-540-89753-8, 978-3-540-89754-5.

[25] John Kelsey, Bruce Schneier e David Wagner – *Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA*. Proceeding ICICS '97 Proceedings of the First International Conference on Information and Communication Security. Springer-Verlag London. 1997. pp 233-246. ISBN: 3-540-63696-X.

[26] Alex Biryukov e Dmitry Khovratovich – *Related-Key Cryptanalysis of the Full AES-192 and AES-256*. Advances in Cryptology – ASIACRYPT 2009. 15th International Conference on the Theory and Application of Cryptology and Information Security,

Tokyo, Japan, December 6-10, 2009. Proceedings. Springer Berlin Heidelberg. 2009. pp 1-18. ISBN: 978-3-642-10365-0, 978-3-642-10366-7.

[27] Jiqiang Lu – *The (related-key) impossible boomerang attack and its application to the AES block cipher*. Designs, Codes and Cryptography. Volume 60. Issue 2. 10-2011. pp 123-143.

[28] Hüseyin Demirci e Ali Aydın Selçuk - *A Meet-in-the-Middle Attack on 8-Round AES*. Fast Software Encryption. 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers. Springer Berlin Heidelberg. 2008. pp 116-126. ISBN: 978-3-540-71038-7, 978-3-540-71039-4.

[29] Hüseyin Demirci, İhsan Taşkın, Mustafa Çoban e Adnan Baysal – *Improved Meet-in-the-Middle Attacks on AES*. Progress in Cryptology - INDOCRYPT 2009. 10th International Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings. Springer Berlin Heidelberg. 2009. pp 144-156. ISBN: 978-3-642-10627-9, 978-3-642-10628-6.

[30] Yongzhuang Wei, Jiqiang Lu e Yupu Hu – *Meet-in-the-Middle Attack on 8 Rounds of the AES Block Cipher under 192 Key Bits*. Information Security Practice and Experience. 7th International Conference, ISPEC 2011, Guangzhou, China, May 30 – June 1, 2011. Proceedings. Springer Berlin Heidelberg. 2011. pp 222-232. ISBN: 978-3-642-21030-3, 978-3-642-21031-0.

[31] Orr Dunkelman, Nathan Keller e Adi Shamir – *Improved Single-Key Attacks on 8-round AES-192 and AES-256*. Advances in Cryptology - ASIACRYPT 2010. 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. Springer Berlin Heidelberg. 2010. pp 158-176. ISBN: 978-3-642-17372-1, 978-3-642-17373-8.

[32] Leibo Li, Keting Jia e Xiaoyun Wang – *Improved Meet-in-the-Middle Attacks on AES-192 and PRINCE*. IACR Cryptology ePrint Archive 2013, 573. 2013.

[33] Patrick Derbez e Pierre-Alain Fouque – *Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES*. Fast Software Encryption. 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Springer Berlin Heidelberg. 2014. pp 541-560. ISBN: 978-3-662-43932-6. 978-3-662-43933-3.

[34] Charles Bouillaguet, Patrick Derbez e Pierre-Alain Fouque – *Automatic Search of Attacks on Round-Reduced AES and Applications*. Advances in Cryptology – CRYPTO 2011. 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Springer Berlin Heidelberg. 2011. pp 169-187. ISBN: 978-3-642-22791-2, 978-3-642-22792-9.

[35] Patrick Derbez, Pierre-Alain Fouque e Jérémy Jean – *Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting*. Advances in Cryptology – EUROCRYPT 2013. 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Springer Berlin Heidelberg. 2013. pp 371-387. ISBN: 978-3-642-38347-2, 978-3-642-38348-9.

[36] Dmitry Khovratovich, Christian Rechberger e Alexandra Savelieva – *Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family*. Fast Software Encryption. 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. Springer Berlin Heidelberg. 2012. pp 244-263. ISBN: 978-3-642-34046-8, 978-3-642-34047-5.

[37] Andrey Bogdanov, Dmitry Khovratovich e Christian Rechberger – *Biclique Cryptanalysis of the Full AES*. Advances in Cryptology – ASIACRYPT 2011. 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. Springer Berlin Heidelberg. 2011. pp 344-371. ISBN: 978-3-642-25384-3, 978-3-642-25385-0.

[38] Biaoshuai Tao e Hongjun Wu – *Improving the Biclique Cryptanalysis of AES*. Information Security and Privacy. 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 -- July 1, 2015, Proceedings. Springer International Publishing. 2015. pp 39-56. ISBN: 978-3-319-19961-0, 978-3-319-19962-7.

[39] Kiran Kumar. V. G., Sudesh Jeevan Mascarenhas, Sanath Kumar, Viven Rakesh J Pais - *Design And Implementation Of Tiny Encryption Algorithm*. International Journal of Engineering Research and Applications. Volume 5. Issue 6. (Parte-2) 06-2015. pp 94-97. ISSN: 2248-9622.

[40] Derek Williams – *The Tiny Encryption Algorithm (TEA)*. CPSC-6128 – Network Security. Columbus State University. 26/04/2008.

[41] Mohammad Shoeb, Vishal Kumar Gupta – *A CRYPT ANALYSIS OF THE TINY ENCRYPTION ALGORITHM IN KEY GENERATION*. International Journal of Communication and Computer Technologies. Volume 01 – No.38. Issue 05. 05-2013. ISSN: 2278-9723.

[42] Sean Murphy – *The cryptanalysis of FEAL-4 with 20 chosen plaintexts*. Artigo do Journal of Cryptology. Volume 2. Issue 3. pp. 145-154. 01-1990. DOI: 10.1007/BF00190801.

[43] Eli Biham e Alex Biryukov – *How to strengthen DES using existing hardware*. Advances in Cryptology — ASIACRYPT'94. 4th International Conferences on the Theory and Applications of Cryptology Wollongong, Australia, November 28 – December 1, 1994 Proceedings. Springer Berlin Heidelberg. 1995. pp 398-412. ISBN: 978-3-540-59339-3, 978-3-540-49236-8.

[44] John Kelsey, Bruce Schneier e David Wagner – *Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES*. Advances in Cryptology — CRYPTO ’96. 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings. Springer Berlin Heidelberg. 1996. pp 237-251. ISBN: 978-3-540-61512-5, 978-3-540-68697-2.

[45] John Kelsey, Bruce Schneier e David Wagner – *Related-key cryptanalysis of 3-WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA*. Information and Communications Security. First International Conference, ICIS '97 Beijing, China, November 11–14, 1997 Proceedings. Springer Berlin Heidelberg.1997. pp 233-246. ISBN: 978-3-540-63696-0, 978-3-540-69628-5.

[46] Dukjae Moon, Kyungdeok Hwang, Wonil Lee, Sangjin Lee e Jongin Lim – *Impossible Differential Cryptanalysis of Reduced Round XTEA and TEA*. Fast Software Encryption. 9th International Workshop, FSE 2002 Leuven, Belgium, February 4–6, 2002 Revised Papers. Springer Berlin Heidelberg. 2002. pp 49-60. ISBN: 978-3-540-44009-3, 978-3-540-45661-2.

[47] Jiazhe Chen, Meiqin Wang e Bart Preneel – *Impossible Differential Cryptanalysis of the Lightweight Block Ciphers TEA, XTEA and HIGHT*. Progress in Cryptology -

AFRICACRYPT 2012. 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings. Springer Berlin Heidelberg. 2012. pp 117-137. ISBN: 978-3-642-31409-4, 978-3-642-31410-0.

[48] Andrey Bogdanov e Meiqin Wang – *Zero Correlation Linear Cryptanalysis with Reduced Data Complexity*. Fast Software Encryption. 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. Springer Berlin Heidelberg. 2012. pp 29-48. ISBN: 978-3-642-34046-8, 978-3-642-34047-5.

[49] Seokhie Hong, Deukjo Hong, Youngdai Ko, Donghoon Chang, Wonil Lee e Sangjin Lee – *Differential Cryptanalysis of TEA and XTEA*. Information Security and Cryptology - ICISC 2003. 6th International Conference, Seoul, Korea, November 27-28, 2003. Revised Papers. Springer Berlin Heidelberg. 2004. pp 402-417. ISBN: 978-3-540-21376-5, 978-3-540-24691-6.

[50] Andrey Bogdanov e Vincent Rijmen – *Zero-Correlation Linear Cryptanalysis of Block Ciphers*. IACR Eprint Archive Report 2011/123. 03/2011.

[51] David Wheeler e Roger Needham – *TEA, a tiny encryption algorithm*. Fast Software Encryption. Second International Workshop Leuven, Belgium, December 14–16, 1994 Proceedings. Springer Berlin Heidelberg. 1995. pp 363-366. ISBN: 978-3-540-60590-4, 978-3-540-47809-6.

[52] David Wheeler e Roger Needham – *Tea extensions*. Computer Laboratory, University of Cambridge. 10-1997.

[53] Gautham Sekar, Nicky Mouha, Vesselin Velichkov e Bart Preneel – *Meet-in-the-Middle Attacks on Reduced-Round XTEA*. Topics in Cryptology – CT-RSA 2011. The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings. Springer Berlin Heidelberg. 2011. pp 250-267. ISBN: 978-3-642-19073-5, 978-3-642-19074-2.

[54] Youngdai Ko, Seokhie Hong, Wonil Lee, Sangjin Lee e Ju-Sung Kang – *Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST*. Fast Software Encryption. 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004. Revised Papers. Springer Berlin Heidelberg. 2004. pp 299-316. ISBN: 978-3-540-22171-5, 978-3-540-25937-4.

[55] David Wagner – *The Boomerang Attack*. Fast Software Encryption. 6th International Workshop, FSE'99 Rome, Italy, March 24–26, 1999 Proceedings. Springer Berlin Heidelberg. 1999. pp 156-170. ISBN: 978-3-540-66226-6, 978-3-540-48519-3.

[56] Eunjin Lee, Deukjo Hong, Donghoon Chang, Seokhie Hong e Jongin Lim – *A Weak Key Class of XTEA for a Related-Key Rectangle Attack*. Progress in Cryptology - VIETCRYPT 2006. First International Conference on Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006. Revised Selected Papers. Springer Berlin Heidelberg. 2006. pp 286-297. ISBN: 978-3-540-68799-3, 978-3-540-68800-6.

[57] Elias Yarrkov – *Cryptanalysis of XXTEA*. Cryptology ePrint Archive, Report 2010/254. 04-05-2010.

[58] ArduinoPortugal.pt – *Arduino Mega*. [Consult. 02/11/2016]. Disponível em http://www.arduinoportugal.pt/arduino-mega/.

[59] ArduinoPortugal.pt – *O que é Internet das Coisas? (Internet of Things [IoT])*. [Consult. 26/12/2016]. Disponível em http://www.arduinoportugal.pt/temas/o-que-e-internet-das-coisas-internet-of-things-iot/

[60] André Silveira de Araujo, Pedro de Vasconcellos – *Bluetooth Low Energy*. Universidade Federal do Rio de Janeiro (UFRJ), Departamento de Engenharia Eletrônica (DEL), Engenharia de Computação e Informação (ECI) – 6º Período, EEL 879 – Redes de Computadores 2 – 2012/2.

[61] Emerson Alecrim – *O que é o Wi-Fi (IEEE 802.11)?*. Publicado em 19-03-2008, atualizado 24-06-2013. [Consult. 15/01/2017]. Disponível em http://www.infowester.com/wifi.php.

[62] Luca Ruggeri – *Top 5 Wireless Ways to Communicate with your Controller*. Publicado em 03-03-2015. [Consult. 15/01/2017]. Disponível em http://www.open-electronics.org/top-5-wireless-ways-to-communicate-with-your-controller/.

[63] Bruna Vasques, Igor Coutinho, Manuela Lima e Vitor Carneval – *ZigBee*. Universidade Federal do Rio de Janeiro, Engenharia Eletrônica e de Computação, Redes de Computadores I – 2010.1. [Consult. 21/01/2017]. Disponível em http://www.gta.ufrj.br/grad/10_1/zigbee/index.html.

[64] open-electronics.org – *MRF24J40MA*. [Consult. 21/01/2017]. Disponível em https://store.open-electronics.org/index.php?_route_=Components/Radio%20Module/MRF24J40MA.

[65] David Kahn – The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet. Publicado em 5 de Dezembro de 1996 pelo Scribner. ISBN: 0684831309.

[66] Eli Biham e Adi Shamir – *Differential Cryptanalysis of the Full 16- round DES*. Differential Cryptanalysis of the Data Encryption Standard. Springer New York. 1993. pp 79-88. ISBN: 978-1-4613-9316-0, 978-1-4613-9314-6.

[67] Whitfield Diffie e Martin E. Hellman – Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. Journal Computer. Volume 10 Issue 6, Junho 1977. pp 74-84. IEEE Computer Society Press Los Alamitos, CA, USA.

[68] MarkT – *AES-library*. [Consult. 10/02/2016]. Disponível em http://utter.chaos.org.uk/~markt/AES-library.zip.

[69] Octoate – *ArduinoDES*. [Consult. 20/02/2016]. Disponível em https://github.com/Octoate/ArduinoDES.

[70] frankmicro – *Xtea*. [Consult. 27/02/2016]. Disponível em https://github.com/franksmicro/Arduino/tree/master/libraries/Xtea.

[71] alessandro1105 – *XXTEA_Arduino_Library*. [Consult. 03/03/2016]. Disponível em https://github.com/alessandro1105/XXTEA_Arduino_Library.

[72] IEEE – *802.11mb Issues List v12* (excel). 20/01/2009. p. CID 98.

[73] Rodrigo R. Paim – *WEP, WAP e EAP*. [Consult. 30/03/2017]. Disponível em https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2011_2/rodrigo_paim/wpa.html.

[74] Carles Gomez, Joaquim Oller e Josep Paradells – *Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology*. Sensors (Basel). 11734–11753. Publicado online em 29/08/2012.

[75] Libelium World – *Security in 802.15.4 and ZigBee networks*. [Consult. 20/04/2017]. Disponível em http://www.libelium.com/security-802-15-4-zigbee/.

Anexo A. Código criado para testar o DES e o TDES

```
#include <DES.h>

DES des;

void setup() {
  Serial.begin(9600);
  Serial.println("Teste DES e 3DES");
  desTeste();
  tdesTeste();
}

void loop() {
}

void desTeste()
{
  byte out[8];
  byte in[] = {
    //password
    0x70, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72,
0x64,
  };
  byte key[] = {
    0x3b, 0x38, 0x98, 0x37, 0x15, 0x20, 0xf7,
0x5e,
  };

  Serial.println();
  Serial.println("========= DES teste
==========");

  //encrypt
  Serial.print("Encriptacao: ");
  long t0 = micros();
  des.encrypt(out, in, key);
  long t1 = micros() - t0;
  Serial.print ("demorou ") ; Serial.print (t1) ;
Serial.println ("us") ;
  printArray(out);

  //decrypt
  for (int i = 0; i < 8; i++)
  {
    in[i] = out[i];
  }
  Serial.print("Desencriptacao: ");
  long t2 = micros();
  des.decrypt(out, in, key);
  long t3 = micros() - t2;
  Serial.print ("demorou ") ; Serial.print (t3) ;
Serial.println ("us") ;
```

```
  printArray(out);
  Serial.print ("Total final: ") ;
  Serial.print ("demorou ") ; Serial.print (t1 +
t3) ; Serial.println ("us") ;

}

void tdesTeste()
{
  byte out[8];
  byte in[] = {
    //password
    0x70, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72,
0x64,
  };
  byte key[] = {
    0x3b, 0x38, 0x98, 0x37, 0x15, 0x20, 0xf7,
0x5e, // key A
    0x92, 0x2f, 0xb5, 0x10, 0xc7, 0x1f, 0x43,
0x6e, // key B
    0x3b, 0x38, 0x98, 0x37, 0x15, 0x20, 0xf7,
0x5e, // key C (in this case A)
  };

  Serial.println();
  Serial.println("====== Triple-DES teste
======");

  //encrypt
  Serial.print("Encriptacao: ");
  long t0 = micros();
  des.tripleEncrypt(out, in, key);
  long t1 = micros() - t0;
  Serial.print ("demorou ") ; Serial.print (t1) ;
Serial.println ("us") ;
  printArray(in);

  //decrypt
  for (int i = 0; i < 8; i++)
  {
    in[i] = out[i];
  }
  Serial.print("Desencriptacao: ");
  long t2 = micros();
  des.tripleDecrypt(out, in, key);
  long t3 = micros() - t2;
  Serial.print ("demorou ") ; Serial.print (t3) ;
Serial.println ("us") ;
  printArray(in);
  Serial.print ("Total final: ") ;
  Serial.print ("demorou ") ; Serial.print (t1 +
t3) ; Serial.println ("us") ;
}
```

```
void printArray(byte output[])
{
  for (int i = 0; i < 8; i++)
  {
    if (output[i] < 0x10)
    {
      Serial.print("0");
    }
    Serial.print(output[i], HEX);
    Serial.print(" ");
  }
  Serial.println();
}
```

Anexo B. Código criado para testar o AES

```
#include <AES.h>

AES aes ;

byte key[] =
{
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
} ;

byte plain[] =
{
  //on
  //0x6f, 0x6e,
  //password
  //0x70, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72,
0x64,
  //0001:password:on
  0x30, 0x30, 0x30, 0x31, 0x3a, 0x70, 0x61, 0x73,
0x73, 0x77, 0x6f, 0x72, 0x64, 0x3a, 0x6f, 0x6e,
} ;

byte my_iv[] =
{
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
} ;

byte cipher [4 * N_BLOCK] ;
byte check [4 * N_BLOCK] ;

void loop ()
{}


void setup ()
{
  Serial.begin (9600) ;
  Serial.println ("Teste AES") ;
  prekey_test () ;

  //  otfly_test () ;
  //  otfly_test256 () ;
}
void testar (int bits, int blocks) {

  byte iv [N_BLOCK] ;

  long t0 = micros () ;
```

```
  byte succ = aes.set_key (key, bits) ;
  long t1 = micros() - t0 ;
  Serial.println () ;
  Serial.print ("set_key ") ; Serial.print (bits)
; Serial.print (" -> ") ;
  Serial.print ("demorou ") ; Serial.print (t1) ;
Serial.println ("us") ;
  long t2 = micros () ;
  if (blocks == 1)
    succ = aes.encrypt (plain, cipher) ;
  else
  {
    for (byte i = 0 ; i < 16 ; i++)
      iv[i] = my_iv[i] ;
    succ = aes.cbc_encrypt (plain, cipher,
blocks, iv) ;
  }
  long t3 = micros ();
  Serial.print ("Encriptacao: ") ;
  Serial.print ("demorou ") ; Serial.print (t3 -
t2) ; Serial.println ("us") ;

  long t4 = micros () ;
  if (blocks == 1)
    succ = aes.decrypt (cipher, plain) ;
  else
  {
    for (byte i = 0 ; i < 16 ; i++)
      iv[i] = my_iv[i] ;
    succ = aes.cbc_decrypt (cipher, check,
blocks, iv) ;
  }

  if (succ != SUCCESS) {
    Serial.print("Erro!");
  }

  long t5 = micros () - t4 ;
  Serial.print ("Desencriptacao: ") ;
  Serial.print ("demorou ") ; Serial.print (t5) ;
Serial.println ("us") ;
  Serial.print ("Total final: ") ;
  Serial.print ("demorou ") ; Serial.print (((t3
- t2) + t1) + t5) ; Serial.println ("us") ;

  for (int i = 0; i < sizeof (plain); i++) {
    Serial.print((char)plain[i]);
  }

  Serial.println () ;

}

void prekey_test ()
```

```
{
  testar(128, 1);
  testar(192, 1);
  testar(256, 1);
}
```

Anexo C. Código criado para testar o XTEA

```
#include <Xtea.h>

unsigned long key[4] = {0x01020304, 0x05060708,
0x090a0b0c, 0x0d0e0f00};
Xtea x(key);

void setup()
{
  Serial.begin(9600);
  unsigned long v[1] = {0x30306F6E};
  long t0 = micros();
  x.encrypt(v);
  long t1 = micros() - t0;
  Serial.print("Encriptacao: ");
  Serial.println(v[0], HEX);
  Serial.print ("demorou ") ; Serial.print (t1) ;
Serial.println ("us") ;
  long t2 = micros();
  x.decrypt(v);
  long t3 = micros() - t2;
  Serial.print("Desencriptacao: ");
  Serial.print(v[0],HEX);
  Serial.print(" ");
  Serial.println(v[1], HEX);
  Serial.print ("demorou ") ; Serial.print (t3) ;
Serial.println ("us") ;
  Serial.print ("Total final: ") ;
  Serial.print ("demorou ") ; Serial.print (t1 +
t3) ; Serial.println ("us") ;
}

void loop()
{
}
```

Anexo D. Código criado para testar o XXTEA

```
#include <XXTEA.h>

#include <Base64.h>

void setup() {

  Serial.begin(9600);

  char s1[] = "password"; //key
  XXTEA xxtea(s1);
  char s[] = "0001passwordon"; //Data
  char output[100];

  Serial.print("Chave: ");
  Serial.println(s1);
  Serial.print("Palavra de entrada: ");
  Serial.println(s);
  Serial.print("Encriptacao: ");
  long t0 = micros();
  xxtea.encrypt(s);
  long t1 = micros() - t0;
  Serial.println(s);
  Serial.print ("demorou ") ; Serial.print (t1) ;
Serial.println ("us") ;

  //base64_decode(output, s, strlen(s));
  //Serial.println(output);
  //Serial.println(strlen(s));

  Serial.print("Desencriptacao: ");
  long t2 = micros();
  xxtea.decrypt(s);
  long t3 = micros() - t2;
  Serial.println(s);
  Serial.print ("demorou ") ; Serial.print (t3) ;
Serial.println ("us") ;
  Serial.print ("Total final: ") ;
  Serial.print ("demorou ") ; Serial.print (t1 +
t3) ; Serial.println ("us") ;

}

void loop() {}
```

Histórico

- 3 de Novembro de 2015, Versão 1.0;
- 9 de Março de 2016, Versão 2.0;
- 27 de Abril de 2016, Versão 3.0;
- 5 de Maio de 2016, Versão 4.0;
- 17 de Junho de 2016, Versão 5.0;
- 18 de Agosto de 2016, Versão 6.0;
- 25 de Setembro de 2016, Versão 7.0;
- 13 de Outubro de 2016, Versão 8.0;
- 29 de Outubro de 2016, Versão 9.0;
- 21 de Janeiro de 2017, Versão 10.0;
- 25 de Fevereiro de 2017, Versão 11.0;
- 01 de Maio de 2017, Versão 12.0;
- 04 de Junho de 2017, Versão 13.0;
- 23 de Junho de 2017, Versão 14.0.