**UNIVERDIDADE PAULISTA**

**APS**

AS TÉCNICAS CRIPTOGRÁFICAS, CONCEITOS, USOS E APLICAÇÕES

**Curso**: Ciência da Computação

**Autores**:

F014BC7 - LUCAS BARBOSA GARCIA

F044289 - ANDREICK GOMES MAIA

F0551I7 - MAICOLN VINICIUS DOS S. SILVA

F107IE2 - ARTHUR RINALDI ZAMPIERI

**SUMÁRIO**

[1. INTRODUÇÃO 2](#_Toc25362480)

[2. OBJETIVO 3](#_Toc25362481)

[3. CRIPTOGRAFIA (CONCEITOS GERAIS) 4](#_Toc25362482)

[4. TÉCNICAS CRIPTOGRÁFICAS MAIS UTILIZADAS E CONHECIDAS 7](#_Toc25362483)

[4.1. CRIPTOGRAFIA ASSIMÉTRICA 7](#_Toc25362484)

[4.1.1. RSA (Rivest-Shamir-Adleman) 8](#_Toc25362485)

[4.2. CRIPTOGRAFIA SIMÉTRICA 9](#_Toc25362486)

[4.2.1. AES (Advanced Encryption Standard) 10](#_Toc25362487)

[4.2.2. Idea (Internacional encryption algorithm) 10](#_Toc25362488)

[5. DISSERTAÇÃO 11](#_Toc25362489)

[6. Estrutura do programa 17](#_Toc25362490)

[6.1. Visão geral do algoritmo 17](#_Toc25362491)

[Gerando as subchaves 18](#_Toc25362492)

[6.2. Algoritmo de encriptação do DES 22](#_Toc25362493)

[6.3. Algoritmo de decriptação do DES 30](#_Toc25362494)

[7. Relatório do código 31](#_Toc25362495)

[7.1. Linhas de código do programa 38](#_Toc25362496)

[8. FICHAS APS 49](#_Toc25362497)

[9. CONCLUSÃO 53](#_Toc25362498)

[Bibliografia 54](#_Toc25362499)

# INTRODUÇÃO

Será tratado sobre os conceitos que envolvem a criptografia, os impactos que causou na história da humanidade. Como toda tecnologia, o recurso vem evoluindo ao longo dos anos e originou diversas ramificações, conhecidas como “chaves”.

As complicações surgem quando se tenta fugir dos limites desses sistemas mais simples.

É preciso desenvolver fórmulas que permitam a uma chave proteger um conteúdo maior que ela própria e criar chaves das quais outras chaves possam ser derivadas para uso temporário ou para que as chaves não tenham de ser transmitidas. Isso é difícil e dá margem para erros que tornam a criptografia não "ideal".

Dessa forma, a criptografia tem como objetivo evitar que terceiros interceptem e desvendem o código.

Além de ser tratado como funcionará a estrutura interna do DES como base para o programa reproduzir a encriptação e decriptação. Em resumo, é o algoritmo mais antigo e "seguro" usado para encriptar textos de tamanho fixo usando uma chave de tamanho fixo.

# OBJETIVO

Este trabalho tem por finalidade abordar de forma ampla os conceitos gerais que envolvem a criptografia, desde a sua concepção até a sua aplicação, expondo questões como a sua relevância para a computação, os impactos que causou na história da humanidade e a sua presença nos mais diversos tipos de tecnologias atualmente, tendo como cenário a rede mundial de computadores.

Será descrito algumas das técnicas de criptografia existentes, em especial as mais conhecidas e utilizadas na informática, explicando suas características gerais, os contextos de aplicação, vantagens e possíveis falhas ou vulnerabilidades, assim como a relação que elas estabelecem entre si.

Também será exposta uma dissertação que apresenta a técnica de criptografia DES, com sua respectiva fundamentação e estruturação, explicação dos conceitos que a envolvem, além de suas características gerais e comparações com as demais técnicas, incluindo todos os outros possíveis tópicos que envolvam a sua aplicação.

Por fim, a conclusão do trabalho implica a elaboração de um projeto em que será utilizada a criptografia DES. O programa desenvolvido deverá efetuar a criptografia e descriptografia de qualquer mensagem, com base em todos os conceitos que serão abordados neste trabalho e no algoritmo da referida técnica criptográfica.

# CRIPTOGRAFIA (CONCEITOS GERAIS)

A palavra criptografia possui sua origem na junção de duas palavras gregas “kryptós” que significa “escondido, oculto”, e “gráphein” que significa “escrita”, portanto, o significado literal da palavra criptografia seria “escrita oculta”. Refere-se ao conjunto de técnicas que são capazes de codificar completamente uma mensagem, onde a interpretação só é possível apenas ao emissor e o receptor, evitando assim que terceiros a decodifiquem. Para que isso ocorra diversas técnicas são utilizadas, e com o passar do tempo são modificas, aperfeiçoadas ou até mesmo substituídas por novas. (Nogueira, 2015)

A criptografia possui quatro objetivos principais, cujo os princípios são:

**confidencialidade**, **integridade**, **autenticação** e **não-repúdio**, ou **irretratabilidade**. Porém nem todos os algoritmos são utilizados a fins de atingir tais objetivos. Esta técnica foi muito utilizada e incrementada durante grandes conflitos, tais como, a Segunda Guerra Mundial ou a Guerra Fria, tentando evitar o vazamento de informações. (Nogueira, 2015)

A necessidade de se enviar e receber informações de maneiras sigilosas já é algo bem antigo, existindo há centenas de anos. Tendo o primeiro uso documentado a cerca de 1900 anos antes de cristo, no Egito, onde foram utilizados hieróglifos (extinto modelo de escrita pictográfica). (Romagnolo, 2007)

O povo hebreu, assim como muitos outros, também utilizavam códigos para transmitir suas mensagens. Onde por meio de uma cifra de substituição simples, monoalfabética e monogâmica, os caracteres podiam ser trocados por outros. (Nogueira, 2015)

Por conta dessa necessidade a criptografia se tornou uma ferramenta essencial para que as mensagens sejam enviadas de forma que somente o emissor e o receptor possuam o acesso livre às informações. (Romagnolo, 2007)

Por meio da criptografia podemos proteger dados sigilosos que estejam armazenados no computador, de forma que utilizada corretamente os mesmos estariam 100% seguros, sendo possível criar uma determinada área no computador onde todas as informações que sejam gravadas ali serão criptografadas automaticamente, tendo possível armazenar arquivos com senhas, e-mails ou qualquer outro tipo de dado pessoal, sendo necessário também proteger os backups contra acessos indevidos, sobretudo aqueles que são enviados para áreas de armazenamento externas, e proteger informações que são realizadas na internet, por exemplo, transações comerciais e bancarias. Já na internet os então chamas de ‘’ sites seguros ‘’ utilizam o protocolo Hyper Text transfer Protocol secure (HTTPS), que seria uma versão do protocolo sob uma camada SSL. Esta camada permite que os dados sejam transmitidos por uma conexão criptografada e que haja a verificação de autenticidade entre o servidor e o cliente através de certificados digitais. (Romagnolo, 2007)

Quando um desses sites seguros é acessado, o navegador utilizado identifica que o protocolo utilizado é o HTTPS, criando assim uma chave negociada diretamente com o servidor de acesso, todo byte de informação que esteja trafegando entre o dispositivo e o servidor é criptografado através desta mesma chave. Assim que os dados são recebidos pelo servidor, ele identifica a criptografia e utiliza a chave para descriptografar, logo após as informações são identificadas, processadas, e criptografadas novamente para enfim serem devolvidas ao navegador.

Já o protocolo HTTP normal não envia as informações de forma criptografada, sendo possível que qualquer monitorador da rede identifique e acesse os dados que estão sendo transmitidos entre o computador e o servidor Web. (Romagnolo, 2007)

A criptografia se resume basicamente em conceitos e técnicas utilizadas para codificar uma informação, de forma que apenas o emissor e o real destinatário da mensagem possam decifra-la, tendo como seu objetivo evitar que terceiros interceptem e compreendam a mensagem.

Atualmente, as técnicas mais famosas são as que envolvem o conceito de chaves criptográficas, contendo um conjunto de bits, onde a quantidade de bits interfere diretamente no nível de segurança da criptografia, esses bits são baseados em um algarismo com a capacidade de interpretar as informações, ou seja, ele é capaz de codificar e decodificar. (Segurança da Informação)

Caso a chave utilizada pelo receptor não seja compatível com a do emissor, as informações não serão extraídas tornando assim impossível sua compreensão.

Neste processo são utilizados 2 tipos de chaves existentes: a chave pública e a chave privada. (Segurança da Informação)

A chave pública é utilizada na codificação das informações, já a chave privada serve para realizar a decodificação. Sendo que a chave pública é acessível a todos, mas para que se possa “abrir” os dados presentes naquela informação, é necessário a utilização da chave privada, cujo o acesso é restrito apenas ao emissor e o receptor original. Com o uso dessas chaves criptográficas, o emissor pode utilizar o mesmo algoritmo para diversos receptores, basta que cada um tenha sua própria chave, sendo possível troca-la de forma que o algoritmo seria mantido. (Segurança da Informação)

Os termos ‘chave de 64 bits’ e ‘chave de 128 bits’ são utilizados para expressar o tamanho das chaves, desta forma, quanto maior for a quantidade de bits utilizados, mais segura será a criptografia. Um exemplo disso seria um algoritmo que utiliza uma chave de 8 bits, por exemplo, somente 256 chaves poderiam ser utilizadas para a decodificação dessa informação, já que 2 elevado a 8 é igual a 256. Desta forma um terceiro poderia gerar 256 tentativas de combinações e tentar decodificar a mensagem, mesmo sendo uma tarefa difícil, não seria algo impossível, portanto, quanto maior a quantidade de bits, mais segura será a criptografia. (Alecrim, 2005)

A criptografia varia de acordo com a aplicação e o nível de segurança requerido, mas no geral, uma criptografia com 56 bits possui seu nível de segurança muito inferior a uma que utiliza 128 bits, por exemplo.

Uma chave de 128 bits possui 339,000,000,000,000,000,000,000,000,000,000,000 de possíveis trocas de caracteres que são utilizados na ocultação da mensagem, já a chave de 56 bits possui 72 quatrilhões de trocas possíveis. Atualmente, o 128 bits é considerado referência de segurança e é utilizado por serviços como Gmail, Facebook e etc. (Garrett, 2012).

# TÉCNICAS CRIPTOGRÁFICAS MAIS UTILIZADAS E CONHECIDAS

## CRIPTOGRAFIA ASSIMÉTRICA

No ano de 1976, Whitfield Diffie e Martin Hellman mudaram os rumos da criptografia assimétrica, também conhecida como criptografia de chave pública, Eles apresentaram um sistema capaz de cifrar e decifrar uma mensagem utilizando duas chaves diferentes, sendo uma delas pública, cujo o acesso era permitido a todos, e a outra privada. Esta criptografia funcionava da seguinte forma: caso a mensagem fosse cifrada com a chave privada ela só poderia ser decifrada com a chave pública e vice-versa. (Nunes, 2007)

O algoritmo presente na chave pública não substitui a criptografia simétrica, por conta de serem lentos e vulneráveis a alguns tipos de ataques. No geral a criptografia da chave pública é utilizada para distribuir de forma segura as chaves simétricas (chaves de seção), visto que esta será utilizada para cifrar as mensagens. Essa técnica funciona de forma que, o emissor gere uma chave simétrica (chave de seção) e esta mesma é cifrada utilizando a chave pública do receptor e a envia, ela só poderá ser decifrada pela pessoa que tiver a chave privada do receptor, na qual somente ele saberá. Desta forma recuperando a chave simétrica e ela assim ela poderá ser utilizada para a comunicação. (Nunes, 2007)

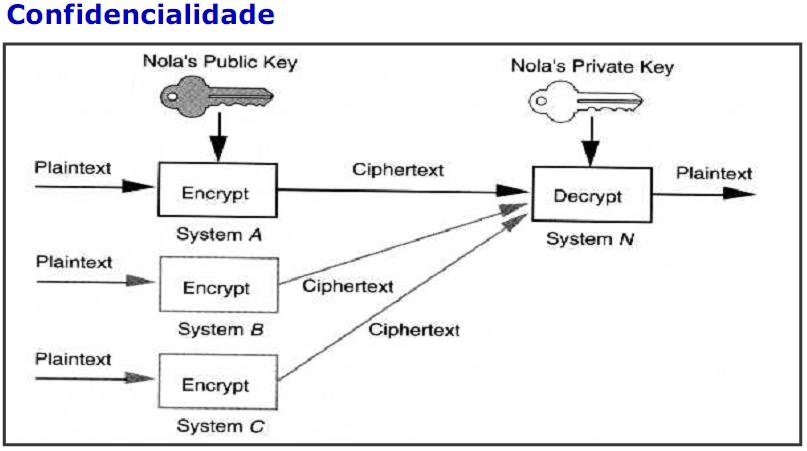


Figura 1 – (Esquema representando o processo de confidencialidade da chave assimétrica.

Fonte: <https://www.gta.ufrj.br/grad/07_2/delio/Criptografiaassimtrica.html>)

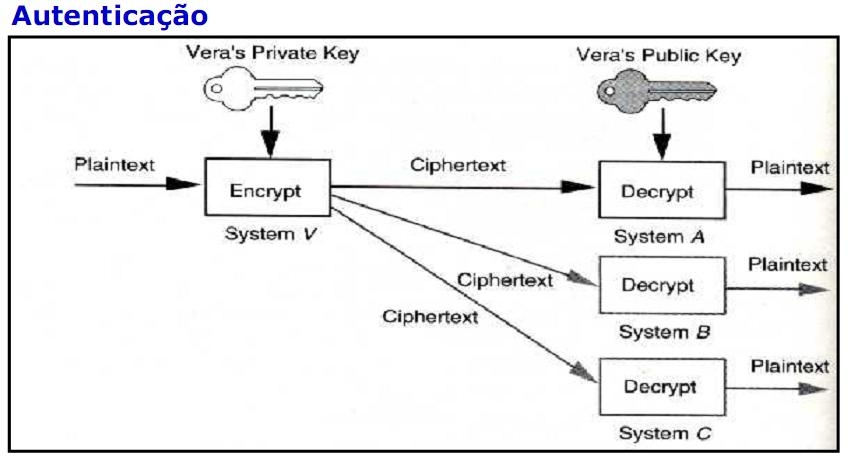


Figura 2 – (Esquema representando o processo de autenticação da chave assimétrica.

Fonte: <https://www.gta.ufrj.br/grad/07_2/delio/Criptografiaassimtrica.html>)

### RSA (Rivest-Shamir-Adleman)

Técnica desenvolvido no Massachussets Institute of Technology (MIT) em 1978 por Rom Rivest, Adi Shamir e Leonard Adleman, que foi batizada com as iniciais de seus nomes. O RSA é baseado matematicamente na Teoria dos Números, sobretudo nas áreas de Aritmética Modular e Primalidade. (Rafael Santos Andrade, 2012)

O conceito do algoritmo RSA dedica-se no fato de que, apesar de ser fácil encontrar dois números primos com grandes dimensões (mais de 100 dígitos), o tempo estimado para fatorar esses números, por exemplo, é de 308 dígitos, com os algoritmos clássicos é de aproximadamente 100 mil anos. Computacionalmente ele se mostra inquebrável com números de tais dimensões, e sua força é calculada com o número de bits utilizados para representar tais números. (Rafael Santos Andrade, 2012)

## CRIPTOGRAFIA SIMÉTRICA

Para se proteger uma determinada informação, garantindo sua privacidade ou confidencialidade, é preciso um algoritmo de criptografia com a capacidade de alterar a mensagem original em uma outra mensagem completamente cifrada, que não seja compreendida por nenhum terceiro. O algoritmo utilizado é chamado de cifra. (Nunes, 2007)

Parte dos métodos criptográficos se baseiam no segredo dos algoritmos. Esses algoritmos possuem somente interesse histórico e não são próprios para as necessidades atuais, visto que se o algoritmo for descoberto será possível abrir qualquer texto cifrado por ele. (Nunes, 2007)

Qualquer algoritmo moderno utiliza uma chave que controla a encriptação, ou seja, uma mensagem só pode ser descriptografada se a chave for a mesma que a utilizada na encriptação, ao contrário da criptografia antiga, onde o segredo estava no algoritmo (Nunes, 2007)

Os algoritmos atuais são projetados para serem executados por computadores e dispositivos específicos. Onde em grande parte das aplicações, a criptografia é feita por um software. No geral, os algoritmos simétricos são executados de forma bem mais rápida do que os assimétricos. (Nunes, 2007)

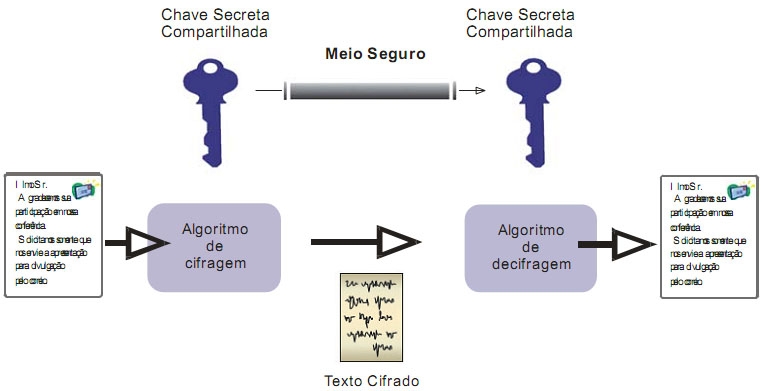


Figura 3 – (Esquema representando o processo de criptografia simétrica.

Fonte: <https://www.gta.ufrj.br/grad/07_2/delio/Criptografiasimtrica.html>)

Podemos perceber que se o receptor e o transmissor tiverem a chave e não a compartilharem com ninguém, apenas eles poderão entender a mensagem, desta forma é garantida a privacidade e a autenticidade da mesma. (Nunes, 2007)

### AES (Advanced Encryption Standard)

O Advanced Encryption Standard (AES) também é chamado de Rijndael cipher. Suportando uma criptografia de 128, 192, 256 bits, que pode ser definida pelo tamanho da chave, a chave de criptografia de 128 bits possui 16 bytes, a chave de 192 bits possui 24 bytes e a de 256 bits possui 32 bytes. A AES fornece um bom desempenho e nível de segurança. Utiliza a cifra simétrica e usa a mesma chave para criptografar e descriptografar. (Soares, 2017)

O Advanced Encryption Standard (AES) talvez seja a criptografia simétrica mais usada no mundo, sendo completamente confiável e compatível com diversos sistemas operacionais. (Morais, 2017)

### Idea (Internacional encryption algorithm)

Elaborada em 1991 o Internacional Encryption Algorithm (IDEA) é uma chave simétrica que opera utilizando blocos de informações de 64 bits e usa chaves de 128 bits. Na prática, o algoritmo atua de forma diferente, porque utiliza a confusão e difusão para cifrar as informações. Ele usa três grupos algébricos com operações misturadas, sendo desta forma cujo o IDEA consegue proteger as informações. (Certificadora, 2019)

# DISSERTAÇÃO

A criptografia é uma tecnologia utilizada há bastante tempo para evitar que terceiros subtraiam ou fraudem informações sigilosas.

É uma ferramenta de segurança amplamente utilizada e consiste basicamente na transformação de determinado dado ou informação a fim de ocultar seu significado real e a DES (Data Encryption Standard) é uma delas.

O DES é um algoritmo para proteger dados usando chaves privadas e simétricas de criptografia desenvolvido no período de 1973 - 1974 baseado num algoritmo mais antigo, o algoritmo Lúcifer de Horst Feistel.

O DES é uma das primeiras criptografias utilizadas e é considerada uma proteção básica de poucos bits. O seu algoritmo é o mais difundido mundialmente e realiza 16 ciclos de codificação para proteger uma informação. O DES é do tipo de cifra em bloco, ou seja, um algoritmo que toma uma string de tamanho fixo de um texto plano e a transforma, através de uma série de complicadas operações, em um texto cifrado de mesmo tamanho, no caso do DES, o tamanho da chave é de 64 bits; porém, somente 56 deles são realmente utilizados pelo algoritmo. Os 8 bits restantes são utilizados para verificar a paridade e integridade e depois são descartados, portanto o tamanho efetivo da chave é de 56 bits.

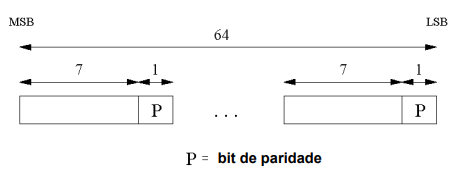


Figura 4 – Divisão de bits da chave (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

Os Blocos que constroem o algoritmo são permutações, substituições, combinações e exclusões. As permutações do DES são de três tipos: na primeira, os bits são simplesmente reordenados (Straight Permutation); na segunda, alguns bits são duplicados e então reordenados (Expanded Permutation), aumentando assim o número de bits na saída; na terceira, alguns bits são descartados para depois reordenar os restantes (Permuted Choice), diminuindo os bits de saída. As substituições no DES são conhecidas como S-Boxes (Caixas S, caixas de substituição). Os ciclos de codificação do DES funcionam como rodadas. Cada rodada usa uma subchave gerada a partir da chave original; as operações matemáticas em cada rodada são as mesmas, a diferença está na subchave e na função ƒ; as funções de expansão, contração e permutação de bits são diferentes para cada rodada; o objetivo das S-box e das permutações é quebrar a linearidade dos dados e a consequência é a dificuldade da inversão da operação de criptografia. O DES também usa uma chave para personalizar a transformação, de modo que a descriptografia somente seria possível, teoricamente, por aqueles que conhecem a chave particular utilizada para criptografar.

A criação do DES trouxe vários benefícios no que diz respeito à segurança da informação, pois uma falha na segurança destas pode acarretar em enormes prejuízos para uma corporação. O DES não vai impedir que uma determinada informação seja interceptada, mas tem o objetivo de dificultar a compreensão. O DES irá modificar a informação de forma que apenas o destinatário consiga compreender o que foi transmitido, garantindo assim, a confidencialidade da informação.

Um outro benefício/vantagem da utilização deste método é a simplicidade na sua implementação, uma vez que é utilizada apenas uma chave no processo de cifragem e decifragem do dado. O método possui um elevado nível de segurança ligado a uma pequena chave para codificação e descodificação, além de ser compreensível, não depender da privacidade do algoritmo, ser adaptável e econômico e ser eficaz. O método ainda apresenta grande velocidade, pois os algoritmos são rápidos, permitindo cifrar uma grande quantidade de dados em pouco tempo.

Este padrão (DES) pode ser usado pelos departamentos e agências federais quando o aplicam-se nas seguintes condições: um funcionário ou gerente autorizado responsável pela segurança dos dados ou pela segurança de qualquer outro sistema de computador decide que a proteção criptográfica é necessária, (bancos, agências internacionais, grandes empresas, contas e transações online...). A criptografia de dados é utilizada em várias aplicações e ambientes. A utilização específica da criptografia e a implementação do DES, será baseado em muitos fatores específicos do sistema de computador e de seus componentes associados. Em geral, a criptografia é usada para proteger os dados enquanto eles estão sendo comunicados entre dois pontos ou enquanto estiver armazenado em um meio vulnerável ao roubo físico. A segurança da comunicação fornece proteção aos dados, codificando-os no ponto de transmissão e decifrando-os no ponto de recebimento. A segurança de arquivos fornece proteção aos dados, codificando-os quando gravados em uma mídia de armazenamento e decifrá-lo quando ele for lido novamente a partir da mídia de armazenamento. No primeiro caso, a chave deve ser disponível no transmissor e no receptor simultaneamente durante a comunicação e no segundo caso, a chave deve ser mantida e acessível durante o período de armazenamento.

O DES ainda possui duas variantes. O 3DES, por exemplo, foi uma alternativa criada para substituir o DES tradicional, mas que, por conta das similaridades entre os dois modelos, se tornou uma segunda opção, e não um substituto de fato. O 3DES trabalha com três chaves de 56 bits cada, criando uma chave de 168 bits no total enquanto o DES tradicional cria apenas uma chave de 56 bits. O DESX, por sua vez, possui os mesmos princípios do DES original, porém é mais resistente a ataques de força bruta – e isso sem tornar o algoritmo mais complexo – ao adicionar 64 bits antes do processo de encriptação, deixando o modelo com 120 bits.

Assim como o DES, o Advanced Encryption Standard (AES) também é uma codificação de bloco de chave simétrica, e foi introduzido para substituir o DES, já que o DES usa uma chave de criptografia muito pequena e o algoritmo é bem mais lento. Uma das diferenças entre o DES e o AES é que o bloco no DES é dividido em duas metades antes do processamento posterior, enquanto no bloco inteiro do AES é processado para obter o texto cifrado. O algoritmo DES trabalha com o princípio Feistel Cipher, já o AES com o princípio de substituição e permutação. Em relação ao tamanho da chave, o DES utiliza uma de 56 bits enquanto o AES pode utilizar uma de 128, 192, ou uma secreta de 256 bits, tornando o AES mais rápido e seguro se comparado ao DES. As rodadas no DES incluem Permutação de Expansão, Xor, S-box, P-box, Xor e Swap. Por outro lado, as rodadas no AES incluem Subbytes, Shiftrows, Mix colunas, Addroundkeys.

Também criado como um substituto ao DES, o Blowfish separa as informações em blocos de 64 bits e criptografa cada um deles de maneira individual. Além disso, o Blowfish tem como destaque a rapidez na encriptação de informações, e é considerado por especialistas um dos poucos modelos que não pode ter o código quebrado.

Criado em 1991, o IDEA é um algoritmo que usa chaves de 128 bits e tem estrutura semelhante ao DES. Essa forma de criptografia opera em blocos de 64 bits e usa chaves de 128 bits, sua principal característica é confundir os atacantes ao misturar as informações codificadas, impedindo que elas sejam realinhadas da maneira correta.

Com a modificação inicial que a chave do DES sofre, transformando-se em duas subchaves que são usadas em partes diferentes do algoritmo, o DES corre o risco de trabalhar com as chamadas chaves fracas. Inicialmente o valor da chave é dividido em duas metades as quais vão sofrer deslocamentos separadamente. Se todos os bits de cada metade forem 0 ou 1, então a chave usada para qualquer ciclo do algoritmo será a mesma usada para qualquer outro ciclo do algoritmo. Isto pode ocorrer se a chave for inteiramente formada por 1 ou inteiramente por 0 ou metade por 1 e metade por 0. Com isso, existem pares de chaves [A, B] onde A, cifra um texto em claro, e tanto A quanto B são capazes de decifrar o criptograma cifrado por A.

Quando a IBM criou o Lúcifer ele tinha uma chave de 128 bits. Alguns anos depois, quando da criação e padronização do DES a chave utilizada para algoritmos criptográficos caiu para 56 bits. Muitos criptólogos da época argumentaram que deveria ser aumentado o número de bits da chave para dificultar a utilização do método da força bruta. Em 1981, dois pesquisadores disseram que, com a evolução da tecnologia computacional, principalmente no que diz respeito a capacidade de armazenamento e de processamento, no ano de 1990 o DES seria um algoritmo completamente inseguro; então, em 1990, uma dupla de israelenses, descobriu e publicou uma técnica nova: a criptoanálise diferencial; entretanto, qualquer algoritmo de criptografia, o método de ataque mais básico é a "força bruta", que é a realização de testes de todas as combinações possíveis. O tamanho da chave determina o número de combinações, e portanto, a dificuldade do acesso. No caso do DES, várias tentativas de quebra (criptoanálise) já foram feitas e publicadas. O DES pode ser quebrado pelo método da "força bruta", tentando-se todas as combinações possíveis para a chave. Como a chave é de 56 bits, tem-se um total de 256 chaves possíveis, ou aproximadamente 1017 possibilidades.

Existem três ataques conhecidos que podem quebrar todos os 16 ciclos do DES com menos complexidade do que a "força bruta", são eles: Criptoanálise Diferencial (Differential Cryptoanalysis); Criptoanálise Linear (Linear Cryptanalysis) e Davies' Attack. Porém, esses ataques são teóricos e não são viáveis na prática. O primeiro ataque conhecido é a Criptoanálise Diferencial, foi redescoberta no final dos anos 1980 pelos israelenses Biham e Shamir, e mantida em segredo pela IBM e NSA. Para quebrar todos os 16 ciclos, a criptoanálise diferencial necessita de 247 textos puros escolhidos (quando o cracker tem a possibilidade de obter textos cifrados a partir de textos puros selecionados). Já o segundo ataque é a Criptoanálise Linear, que foi descoberta por Mitsuru Matsui, e precisa de 243 textos puros conhecidos. O método foi implementado por Matsui em 1994, e foi a primeira experiência de criptoanálise do DES a ser reportada, não existe evidência que o DES foi desenvolvido para ser resistente a este tipo de ataque. Uma generalização da Criptoanálise Linear Múltipla, foi sugerida em 1994 por Kaliski e Robshaw e depois, aperfeiçoada por Biryukov em 2004. Análises sugerem que aproximações Lineares Múltiplas podem ser usadas para reduzir em 4 vezes o número total de dados necessários para o ataque (241 ao invés de 243). Junod, em 2001, realizou vários experimentos para determinar o tempo atual da complexidade da Criptoanálise Linear. O terceiro ataque é o Improved Davies' Attack, que enquanto a criptoanálise diferencial e linear são técnicas genéricas e podem ser aplicadas em diferentes exemplos, o Davies' Attack é uma técnica especializada em DES, sugerida primeiramente por Donald Davies nos anos 1980, e aperfeiçoado em 1997. A forma mais poderosa do ataque necessita 250 textos puros conhecidos, tem uma complexidade computacional de 250, e 51% de sucesso.

Embora o DES tenha trazido vários benefícios no que diz respeita à segurança de informações melhorias tiveram que ser feitas para garantir a confidencialidade de informações já que ao longo do tempo a tecnologia também evoluiu. O uso de algoritmos mais complexos e capazes de criar obstáculos grandes tornou - se essencial. Como propostas de melhorias temos o 3DES (Triplo DES), AES(Advanced Encryption Standard).

O 3DES, sigla para Triple Data Encryption Standard, é um padrão de criptografia baseado em outro algoritmo de criptografia simétrica, o DES, desenvolvido pela IBM em 1974 e adotado como padrão em 1977. O 3DES usa 3 chaves de 64 bits, embora apenas 56 bits de cada chave são efetivamente usados, os outros 8 bits são usados para verificar paridade. Sendo assim, o tamanho máximo efetivo da chave é de 168 bits. Os dados são encriptados com a primeira chave, decriptados com a segunda chave e finalmente encriptados novamente com uma terceira chave. Isto faz o 3DES ser mais lento que o DES original, porém em contrapartida oferece maior segurança. Hoje, distinguimos vários tipos de codificação Triplo DES: DES-EEE3: três codificações DES com 3 chaves diferentes; DES-EDE3: uma chave diferente para cada uma das 3 operações DES (codificação, descodificação, codificação); DES-EEE2 e DES-EDE2: uma chave diferente para a segunda operação (descodificação).

Ainda como melhoria de proteção de dados foi criado o AES, um algoritmo de codificação destinado a substituir o DES por acreditarem não ser tão seguro.

# Estrutura do programa

A estrutura interna do DES serviu como base para o programa reproduzir a encriptação e decriptação, apenas a maneira como as subchaves são produzidas difere de sua estrutura original, este detalhe assim como a explicação do algoritimo dessa criptografia e o respectivo código do programa serão abordados a seguir.

## Visão geral do algoritmo

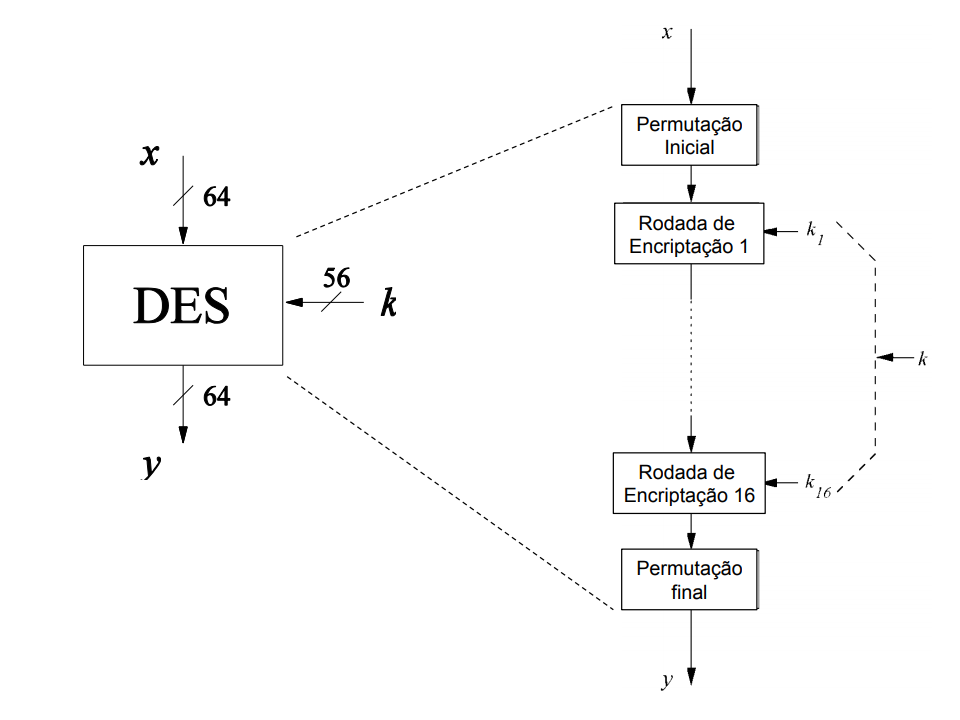
****

Figura 5 - Fluxograma da cifra de bloco DES (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

O DES criptografa blocos de 64 bits usando uma chave de 56 bits e 16 rodadas de encriptação, todas com operações idênticas e subchaves diferentes, produzindo ao final um bloco de 64 bits criptografado.

As permutações são a troca da posição dos bits utilizando um vetor, representado em forma de tabela para facilitar a apresentação, que utiliza uma numeração em ordem crescente da posição inicial dos bits, começando com msb (most significant bit) na posição um. A tabela apresentada mostra onde cada bit deve estar, ou seja, a posição do número 1 da tabela representa a posição do bit de posição inicial um após a permutação.

## Gerando as subchaves

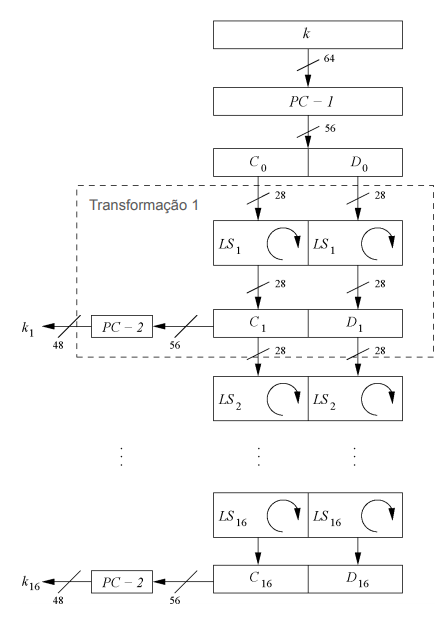


Figura 6 - Fluxograma do algoritmo da formação de subchaves (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

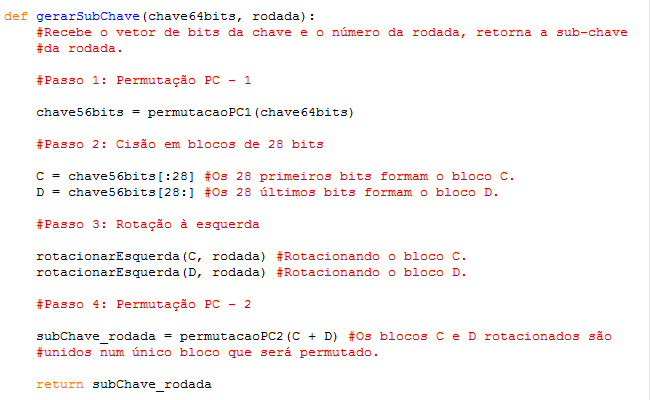


Figura 7 - Algoritmo da formação de subchaves em Python

A partir da chave original de 56 bits são obtidas 16 sub-chaves de 48 bits. A primeira etapa desse processo é a permutação com seleção PC – 1.

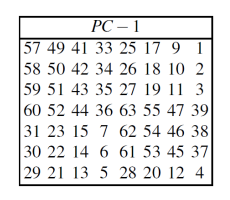


Figura 8 – Permutação PC-1 (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

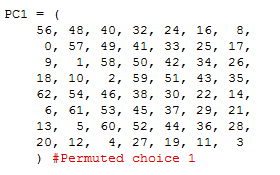


Figura 9 – Permutação PC-1 em Python

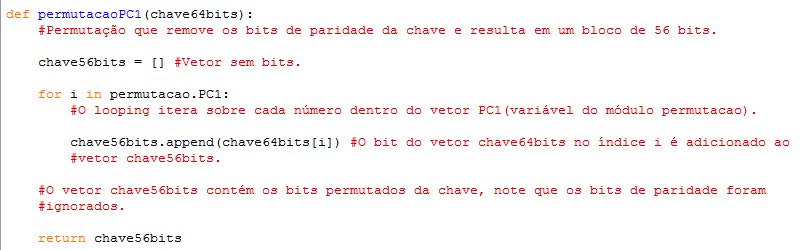


Figura 10 – Código em Python que realiza a permutação PC-1

Note que esta permutação é a responsável por remover os bits de paridade, produzindo um bloco de 56 bits permutado. Na segunda etapa esse bloco é dividido em duas metades de 28 bits nomeadas de C0 e D0, ambas são rotacionadas à esquerda, o que significa trocar a posição de todos os bits do bloco igualmente, movendo cada um para a esquerda, o bit que estiver na primeira posição do bloco vai para a última posição.

A rotação é o procedimento que torna as subchaves diferentes entre si, pois da segunda rodada em diante utiliza-se os blocos C e D da subchave da rodada anterior para gerar uma nova subchave, o número de rotações à esquerda é pré-definido e varia entre 1 e 2 para cada rodada.

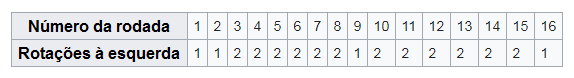


Figura 11 – Tabela do algoritmo com o número de rotações da respectiva rodada (Wikipedia, the free encyclopedia, 2019)

No programa os blocos C0 e D0 são rotacionados para formar cada uma das 16 subchaves, sem precisar usar a subchave anterior, portanto o número de rotações aumenta conforme a progressão das rodadas, conforme a tabela a seguir.

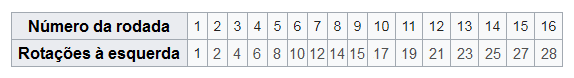


Figura 12 – Tabela usada no programa com o número de rotações da respectiva rodada

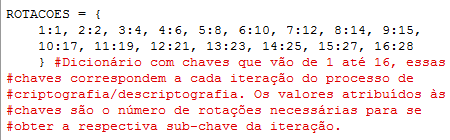


Figura 13 – Variável do módulo permutacao para fazer as rotações do programa

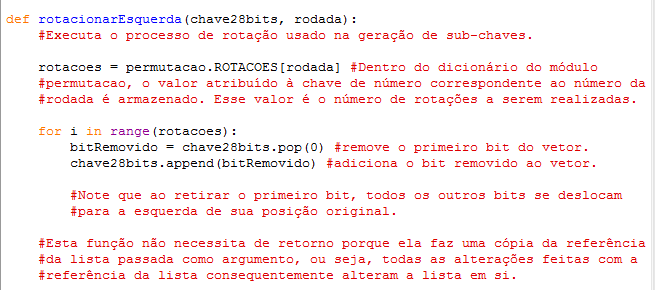


Figura 14 – Código em Python que realiza as rotações

A última etapa consiste em unir os dois blocos rotacionados formando um bloco de 56 bits que passa pela permutação com seleção PC – 2, resultando na subchave da rodada.

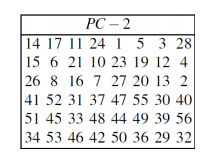


Figura 15 – Permutação PC-2 (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

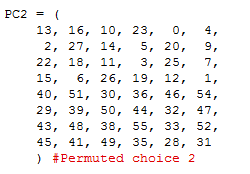


Figura 16 – Permutação PC-2 em Python

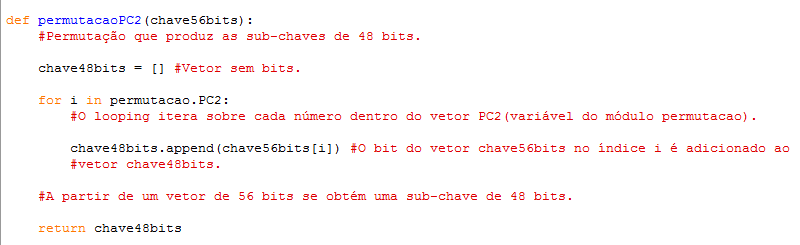


Figura 17 - Código em Python que realiza a permutação PC-2

## Algoritmo de encriptação do DES

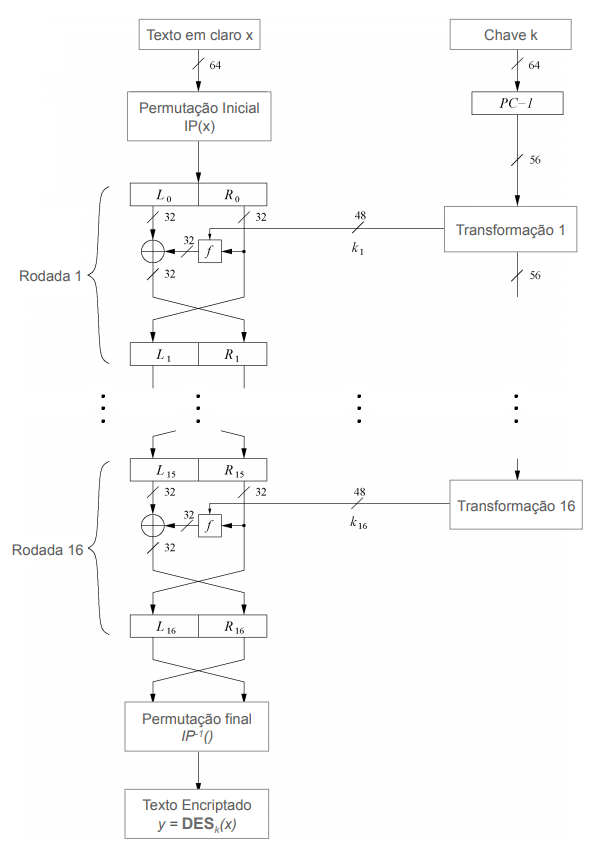


Figura 18 - Fluxograma do algoritmo de encriptação (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

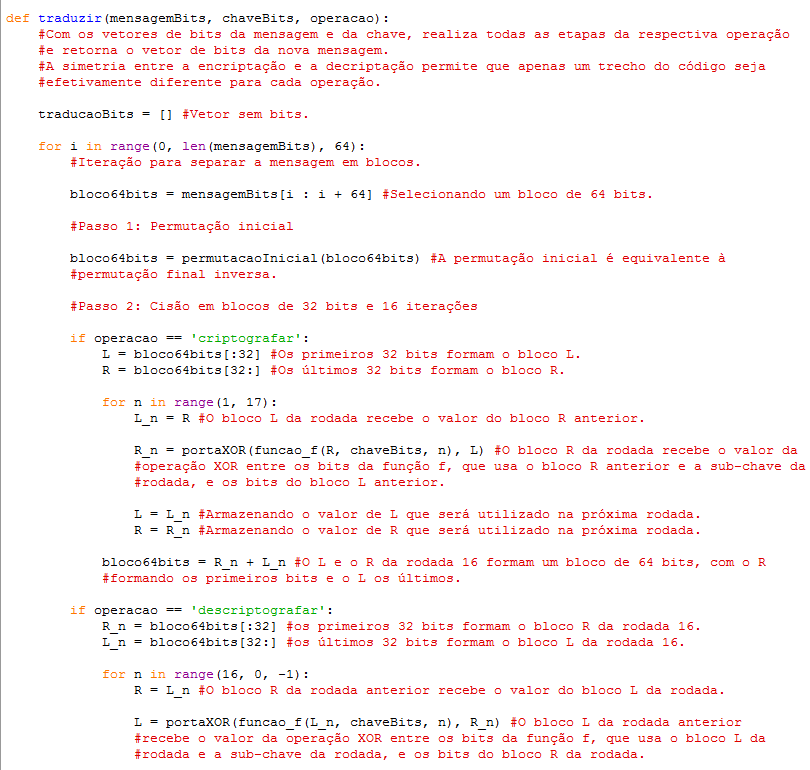


Figura 19 – Algoritmo de encriptação em Python

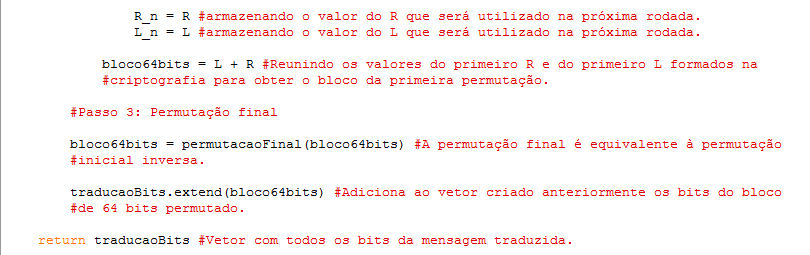


Figura 20 - Algoritmo de encriptação em Python

A função acima também faz a decriptação, a string armazenada na variável operacao é responsável por alternar entre os dois procedimentos.

Depois de fracionado o texto puro, o primeiro procedimento do algoritmo é passar um bloco de 64 bits pela permutação inicial.

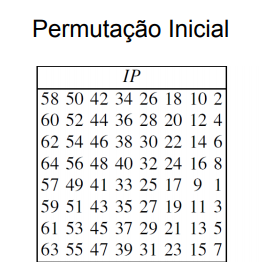


Figura 21 – Permutação inicial (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

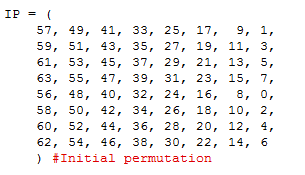


Figura 22 – Permutação inicial em Python

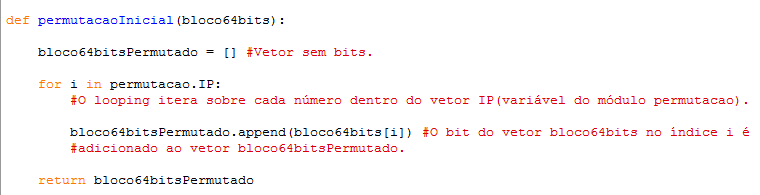


Figura 23 - Código em Python que realiza a permutação inicial

O bloco permutado é dividido dando origem ao bloco esquerdo L0 e ao bloco direito R0, ambos de 32 bits. Aqui começam as 16 transformações iterativas, na primeira rodada o bloco L1 recebe o bloco R0 e para obter o bloco R1 é usado o bloco L0 e a função ƒ, principal operação do DES, detalhada a seguir.

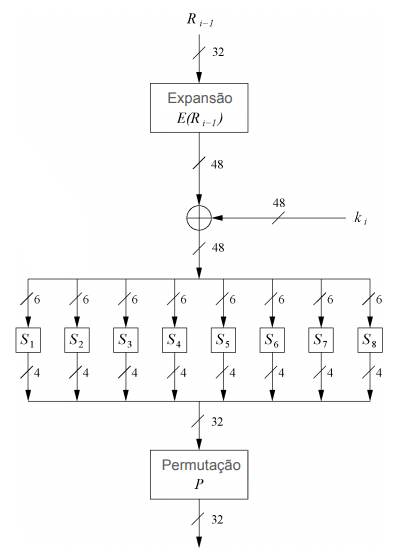


Figura 24 – Fluxograma do algoritmo da função ƒ (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

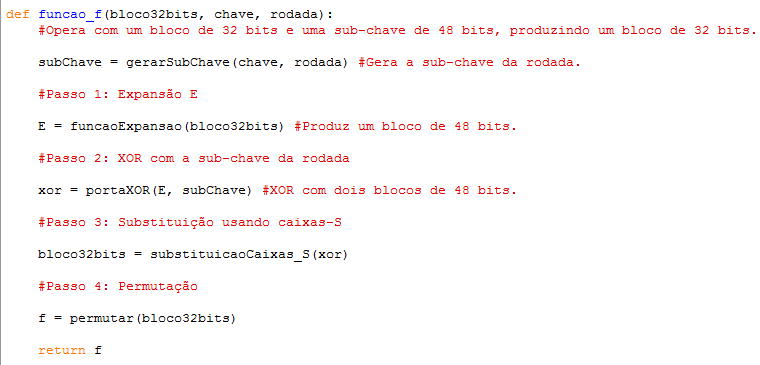


Figura 25 – Algoritmo da função ƒ em Python

A função ƒ opera em um bloco de 32 bits com uma subchave e produz um bloco de 32 bits. A primeira etapa é a função de expansão E.

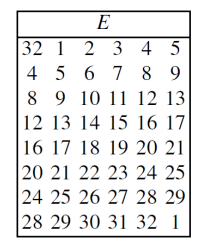


Figura 26 – Expansão E (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

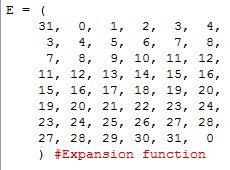


Figura 27 – Expansão E em Python

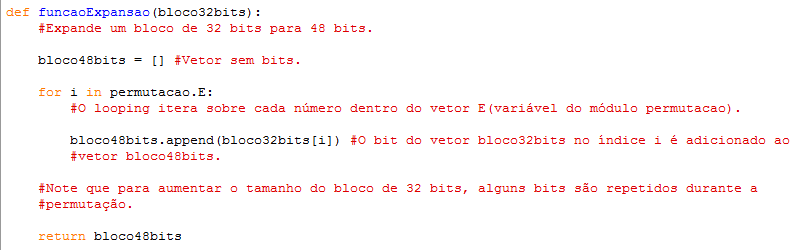


Figura 28 - Código em Python que realiza a expansão E

Com a expansão E o bloco de 32 bits é estendido para 48 bits, em seguida ele passa por um XOR bit a bit com a subchave da rodada, procedimento também chamado de soma XOR que aplica essa operação lógica com bits de cada bloco produzindo um novo bloco de 48 bits.

A próxima etapa envolve as caixas S do DES, são oito tabelas de substituição com linhas e colunas que operam com uma fração de 6 bits do bloco derivado da soma XOR. Os bits servem de endereço para as tabelas.

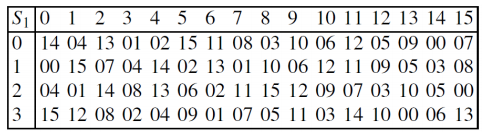


Figura 29 – Caixa S1 (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

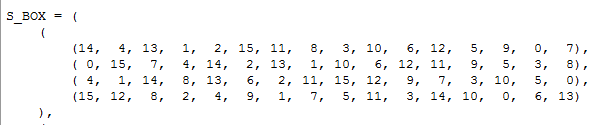


Figura 30 – Caixa S1 em Python

Dos 6 bits fracionados são extraídos dois números binários, um usando o primeiro e o último bit e o outro usando os 4 bits restantes, que convertidos para decimal definem respectivamente a linha e a coluna da caixa S.

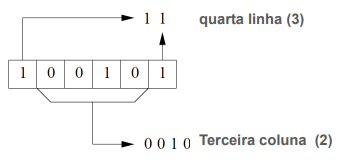


Figura 31 – Exemplo de divisão dos bits para selecionar a linha e a coluna das caixas S (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

O número da linha e coluna definidas é convertido para binário resultando em 4 bits. Reunindo os oito blocos de 4 bits obtidos forma-se um bloco de 32 bits.

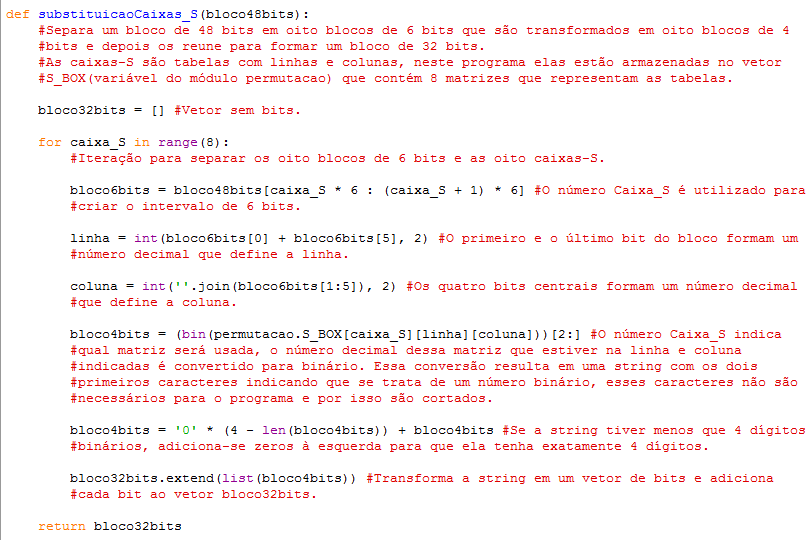


Figura 32 – Substituição das caixas S em Python

A última etapa da função ƒ é a permutação P cuja saída é um bloco de 32 bits.

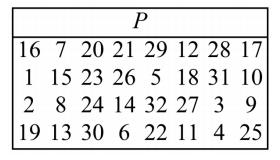


Figura 33 – Permutação P (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

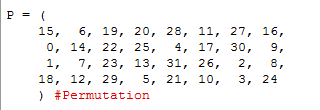


Figura 34 – Permutação P em Python

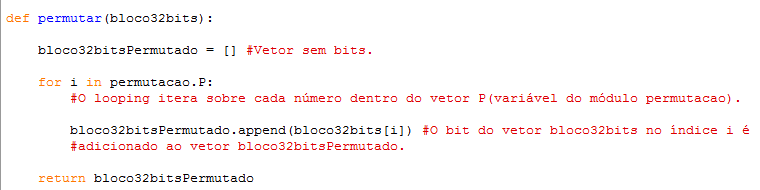


Figura 35 - Código em Python que realiza a permutação P

Retomando a obtenção do R1, a função ƒ recebe o bloco R0 e a chave da rodada retornando um bloco de 32 bits que, através da soma XOR com o bloco L0, resulta no bloco R1.

Nas próximas iterações são feitas as mesmas operações, o bloco L da rodada recebe o bloco R da rodada anterior enquanto o bloco R da rodada recebe a soma XOR entre o bloco L anterior e a saída da função f, que usa o R anterior e a chave da rodada. Ao final das iterações teremos os blocos L16 e R16, a ordem deles é invertida para agrupa-los em um bloco de 64 bits.

O ultimo procedimento do algoritmo é a permutação final cuja saída é um bloco de 64 bits do texto cifrado.

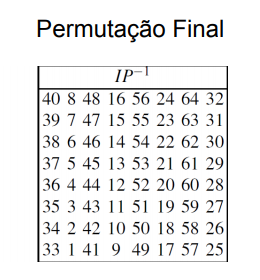


Figura 36 – Permutação final (Pelzl & Paar, ch03-DES-slides-pt.pdf, 2010)

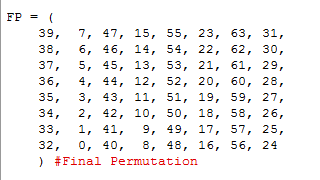


Figura 37 – Permutação final em Python

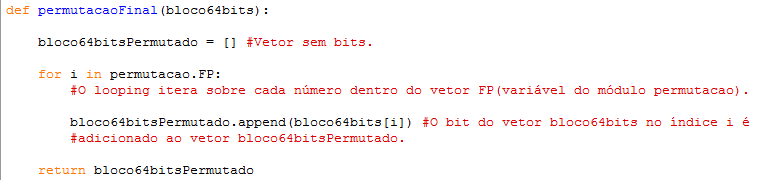


Figura 38 - Código em Python que realiza a permutação final

## Algoritmo de decriptação do DES

Na decriptação do DES todas as etapas da encriptação são desfeitas, tornando os algoritmos muito semelhantes, a diferença entre eles está na ordem das 16 rodadas que é invertida na decriptação, ou seja, a primeira rodada é a 16 e a última rodada é a 1, revertendo os blocos L16 e R16 para os blocos L0 e R0.

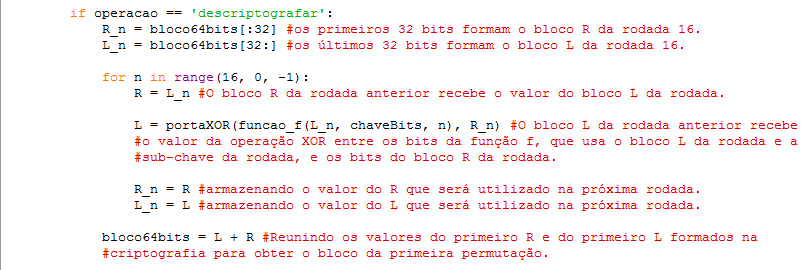


Figura 39 – Trecho do código de decriptação que difere do código de encriptação

# Relatório do código

Além das funções que fazem a encriptação e decriptação, o programa também tem funções de comunicação com o usuário e funções de manipulação e conversão de bits.

A criptografia DES trabalha com blocos de bits, entretanto o usuário deve digitar uma mensagem do tipo caractere ou hexadecimal e uma chave hexadecimal, por isso o programa tem funções que convertem esses valores para bits e vice-versa, isso é possível utilizando a tabela ASCII. Um byte corresponde a um número hexadecimal de dois dígitos e a um caractere. Vale destacar que todos esses valores usados no programa são representados por strings, não é necessário utilizar o tipo equivalente ao valor.

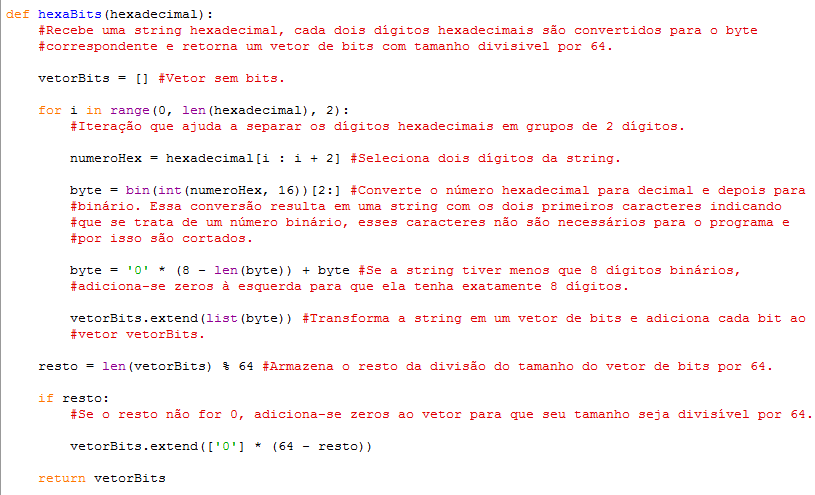


Figura 40 – Conversão de hexadecimal para bits em Python

A função acima converte cada dois dígitos do número hexadecimal passado para o byte correspondente, a saída é um vetor com cada bit. Tanto a mensagem quanto a chave passam por essa função para se obter os bits, como a mensagem será quebrada em blocos de 64 bits a função também garante que o vetor de bits tenha exatamente 64 bits adicionando bits 0 ao vetor, isto permite que a mensagem ao ser dividida possua o último bloco completo sem interferir na compreensão da mensagem. No caso da chave os bits 0 devem ficar na frente do número hexadecimal para não alterar o valor da chave, o que deve ser feito antes que ela seja enviada para a função, o código que faz isso será mostrado mais adiante.

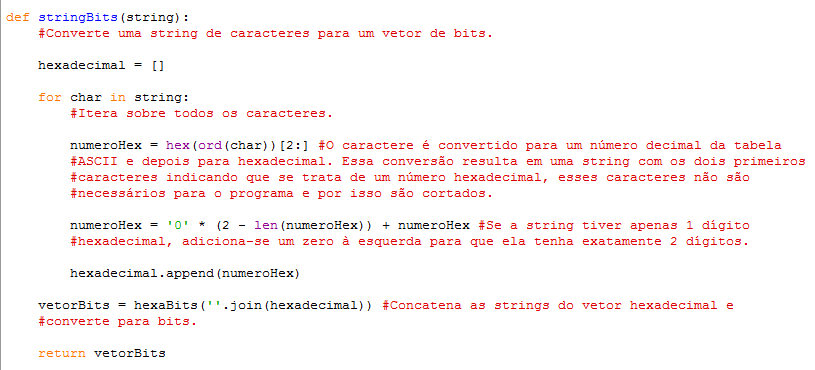


Figura 41 - Conversão de string para bits em Python

Esta função é utilizada para converter a mensagem para bits se ela for do tipo caractere, primeiro os caracteres são convertidos para o número hexadecimal correspondente, depois a função hexaBits se encarrega de terminar a conversão.

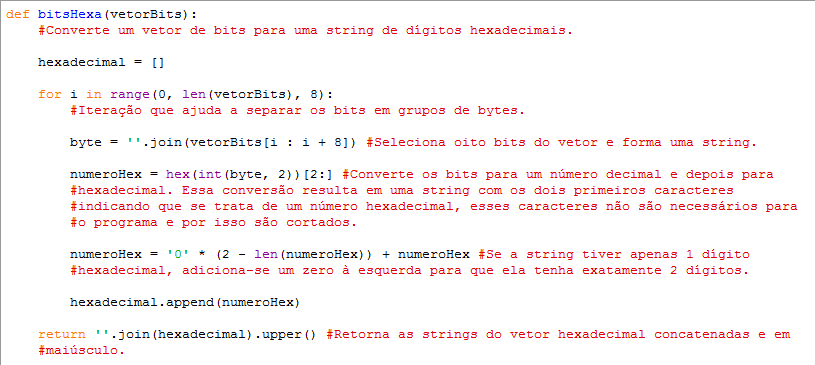


Figura 42 - Conversão de bits para hexadecimal em Python

A função acima faz o inverso da função hexaBits, cada 8 bits de um vetor são convertidos para o número hexadecimal correspondente, a saída é uma string com todos os dígitos hexadecimais derivados da conversão.

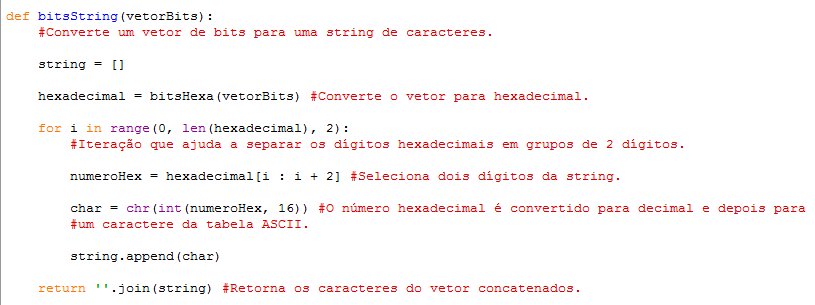


Figura 43 - Conversão de bits para string em Python

Esta função faz a conversão inversa feita por stringBits de maneira análoga, primeiramente chama a função bitsHexa para converter os bits para hexadecimal e depois converte cada dois dígitos hexadecimais para o caractere correspondente.

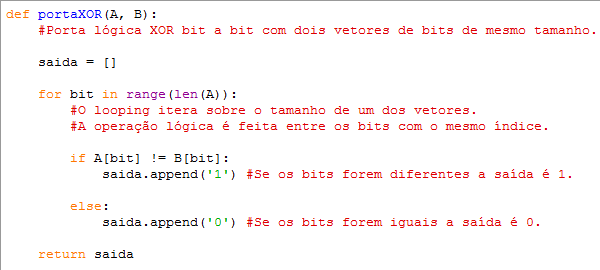


Figura 44 – Porta XOR em Python

Esta função emula a porta lógica XOR utilizada duas vezes na criptografia.

A seguir estão as funções de interação humano-máquina.

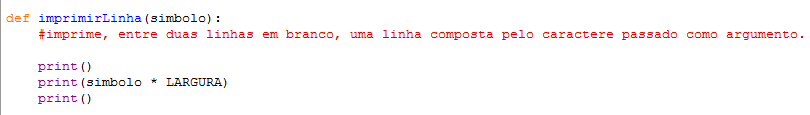


Figura 45 – Código em Python para imprimir uma linha de caracteres

A função acima imprime uma linha constituída de caracteres iguais, a quantidade é definida pela variável LARGURA que delimita o tamanho da interface do programa. A intenção é utilizar essas linhas de caracteres para ajudar a organizar as informações exibidas ao usuário.

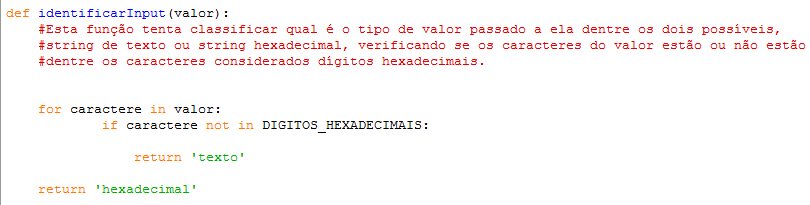


Figura 46 – Código em Python para classificar o tipo de um valor em texto ou hexadecimal

A função acima identifica qual é o tipo de valor representado pela string passada, ela opera com as entradas do usuário cujo tipo pode ser caractere ou hexadecimal. Iterando sobre cada caractere da string, a função só define o tipo como hexadecimal se todos os caracteres forem dígitos hexadecimais, os dígitos aceitos

estão na constante DIGITOS\_HEXADECIMAIS, se um caractere não estiver na constante então o valor é do tipo é caractere. A função pode não funcionar em todos

os casos, mas para o objetivo do programa o resultado é satisfatório.

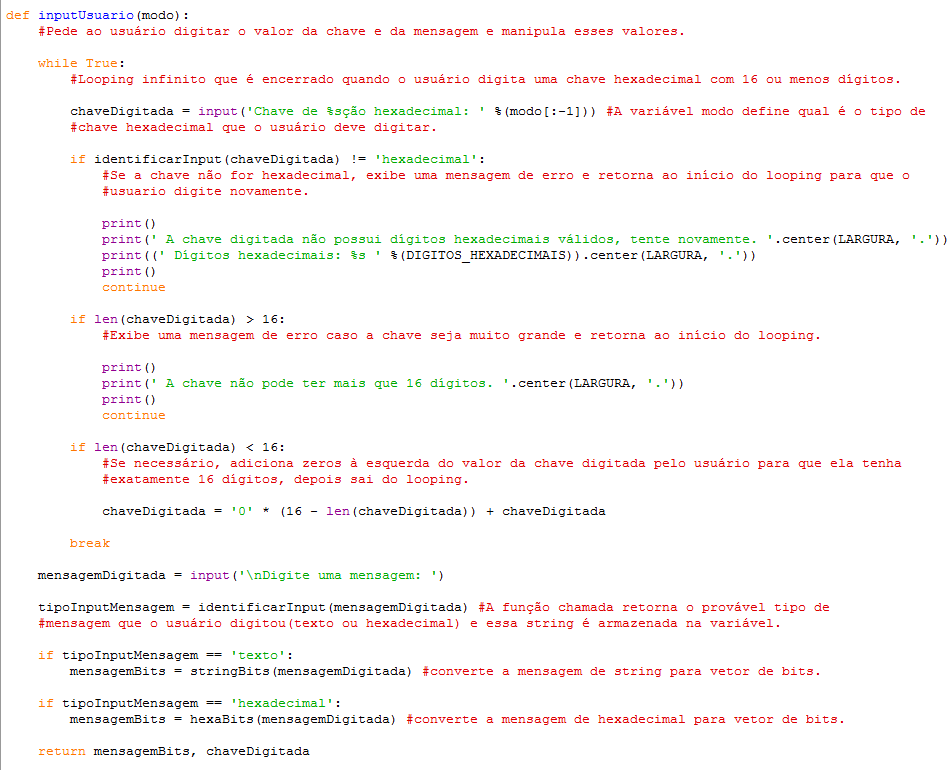


Figura 47 – Código do programa que gerencia a mensagem e a chave

Esta função é responsável por pedir e manipular a chave e a mensagem digitadas pelo usuário, primeiro é solicitado a chave, caso ela não seja hexadecimal ou tenha mais que 16 dígitos (mais que 64 bits) uma mensagem de erro é exibida e o programa pede a chave novamente, aqui também é feita a adição de zeros em frente ao número hexadecimal caso o tamanho não seja adequado, se a chave não tiver 16 dígitos, quando ela for passada para a função hexaBits o seu valor será alterado com a adição dos bits 0, o que deve ser evitado.

Depois é solicitado a mensagem que será criptografada/descriptografada e de acordo com o retorno da função identificarInput ela é convertida de caractere para bits ou de hexadecimal para bits.

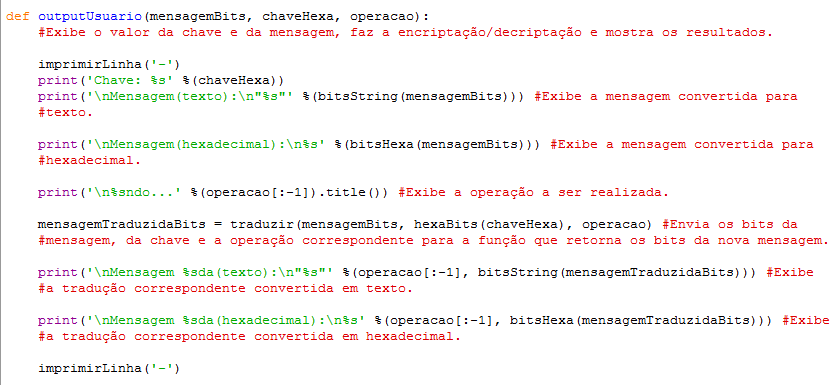


Figura 48 – Código do programa que exibe informações da  
 criptografia ou descriptografia para o usuário

A função acima exibe ao usuário a chave, já com os 16 dígitos, e a mensagem em formato de texto e de número hexadecimal, depois faz a operação que o usuário escolheu (criptografar ou descriptografar) exibindo a mensagem produzida também nos formatos de texto e de número hexadecimal.



Figura 49 – Código da função main

A função main integra em seu código todos os processos realizados pelo programa e o direcionamento das variáveis para as respectivas funções. Primeiro ela exibe o título e o menu do programa, este que está dentro de um looping infinito que se encerra quando o usuário escolhe a opção “Sair”, permitindo que o usuário faça as operações possíveis quantas vezes desejar.

Ao pedir que o usuário escolha uma opção, se ela não for válida uma mensagem de erro é exibida, se for escolhida uma operação são definidas as variáveis que o programa deve utilizar. A string modoImpressaoChave é passada para a função inputUsuario que pede ao usuário a chave da operação correspondente, enquanto a string operacao é passada para a função outputUsuario que a utiliza na função traduzir, que de acordo com essa variável faz a criptografia ou descriptografia, e também a utiliza para exibir as mensagens ao usuário condizentes com operação escolhida.

# Linhas de código do programa

#Criptografia DES

import permutacao #Módulo importado para dar suporte às funções que realizam as substituiçõs e

#permutações dos blocos de bits.

#Constantes:

LARGURA = 80 #Define o valor tido como parâmetro de largura para a tela do usuário.

DIGITOS\_HEXADECIMAIS = '0123456789ABCDEF' #Define os dígitos hexadecimais aceitos pelo #programa.

#Funções de interação com o usuário:

def main():

print()

print('\*' \* LARGURA)

print('CRIPTOGRAFIA DES'.center(LARGURA))

print()

print('Modo ECB(Eletronic CodeBook)'.center(LARGURA))

print('\*' \* LARGURA)

print()

while True:

#Looping infinito que encerra quando o usuário escolher sair.

print('Digite uma opção:')

print('\t1 - Criptografar')

print('\t2 - Descriptografar')

print('\t3 - Sair')

opcao = input('Opção: ') #A escolha determina se o programa faz uma operação ou se ele é

#encerrado.

if opcao == '1':

imprimirLinha('-')

modoImpressaoChave = 'encriptar'

operacao = 'criptografar'

elif opcao == '2':

imprimirLinha('-')

modoImpressaoChave = 'decriptar'

operacao = 'descriptografar'

elif opcao == '3':

imprimirLinha('-')

print('Programa encerrado.')

print()

break

else:

#Se o usuário não digitar nenhuma das três opções é exibida uma mensagem de erro e o

#looping volta ao início.

print()

print(' Opção inválida, tente novamente. '.center(LARGURA, '-'))

print()

continue

mensagemBits, chaveHexa = inputUsuario(modoImpressaoChave)

outputUsuario(mensagemBits, chaveHexa, operacao)

def imprimirLinha(simbolo):

#imprime, entre duas linhas em branco, uma linha composta pelo caractere passado como argumento.

print()

print(simbolo \* LARGURA)

print()

def inputUsuario(modo):

#Pede ao usuário digitar o valor da chave e da mensagem e manipula esses valores.

while True:

#Looping infinito que é encerrado quando o usuário digita uma chave hexadecimal com 16 ou menos dígitos.

chaveDigitada = input('Chave de %sção hexadecimal: ' %(modo[:-1])) #A variável modo define qual é o tipo de chave hexadecimal que o usuário deve digitar.

if identificarInput(chaveDigitada) != 'hexadecimal':

#Se a chave não for hexadecimal, exibe uma mensagem de erro e retorna ao início do looping para que o usuario digite novamente.

print()

print(' A chave digitada não possui dígitos hexadecimais válidos, tente novamente. '.center(LARGURA, '.'))

print((' Dígitos hexadecimais: %s ' %(DIGITOS\_HEXADECIMAIS)).center(LARGURA, '.'))

print()

continue

if len(chaveDigitada) > 16:

#Exibe uma mensagem de erro caso a chave seja muito grande e retorna ao início do looping.

print()

print(' A chave não pode ter mais que 16 dígitos. '.center(LARGURA, '.'))

print()

continue

if len(chaveDigitada) < 16:

#Se necessário, adiciona zeros à esquerda do valor da chave digitada pelo usuário para que ela tenha exatamente 16 dígitos, depois sai do looping.

chaveDigitada = '0' \* (16 - len(chaveDigitada)) + chaveDigitada

break

mensagemDigitada = input('\nDigite uma mensagem: ')

tipoInputMensagem = identificarInput(mensagemDigitada) #A função chamada retorna o provável tipo de mensagem que o usuário digitou(texto ou hexadecimal) e essa string é armazenada na variável.

if tipoInputMensagem == 'texto':

mensagemBits = stringBits(mensagemDigitada) #converte a mensagem de string para vetor de bits.

if tipoInputMensagem == 'hexadecimal':

mensagemBits = hexaBits(mensagemDigitada) #converte a mensagem de hexadecimal para vetor de bits.

return mensagemBits, chaveDigitada

def identificarInput(valor):

#Esta função tenta classificar qual é o tipo de valor passado a ela dentre os dois possíveis,

#string de texto ou string hexadecimal, verificando se os caracteres do valor estão ou não estão

#dentre os caracteres considerados dígitos hexadecimais.

for caractere in valor:

if caractere not in DIGITOS\_HEXADECIMAIS:

return 'texto'

return 'hexadecimal'

def outputUsuario(mensagemBits, chaveHexa, operacao):

#Exibe o valor da chave e da mensagem, faz a encriptação/decriptação e mostra os resultados.

imprimirLinha('-')

print('Chave: %s' %(chaveHexa))

print('\nMensagem(texto):\n"%s"' %(bitsString(mensagemBits))) #Exibe a mensagem convertida para texto.

print('\nMensagem(hexadecimal):\n%s' %(bitsHexa(mensagemBits))) #Exibe a mensagem convertida para hexadecimal.

print('\n%sndo...' %(operacao[:-1]).title()) #Exibe a operação a ser realizada.

mensagemTraduzidaBits = traduzir(mensagemBits, hexaBits(chaveHexa), operacao) #Envia os bits da mensagem, da chave e a operação correspondente para a função que retorna os bits da nova mensagem.

print('\nMensagem %sda(texto):\n"%s"' %(operacao[:-1], bitsString(mensagemTraduzidaBits))) #Exibe a tradução correspondente convertida em texto.

print('\nMensagem %sda(hexadecimal):\n%s' %(operacao[:-1], bitsHexa(mensagemTraduzidaBits))) #Exibe a tradução correspondente convertida em hexadecimal.

imprimirLinha('-')

#Funções de criptografia/descriptografia:

def traduzir(mensagemBits, chaveBits, operacao):

#Com os vetores de bits da mensagem e da chave, realiza todas as etapas da respectiva operação

#e retorna o vetor de bits da nova mensagem.

#A simetria entre a encriptação e a decriptação permite que apenas um trecho do código seja

#efetivamente diferente para cada operação.

traducaoBits = [] #Vetor sem bits.

for i in range(0, len(mensagemBits), 64):

#Iteração para separar a mensagem em blocos.

bloco64bits = mensagemBits[i : i + 64] #Selecionando um bloco de 64 bits.

#Passo 1: Permutação inicial

bloco64bits = permutacaoInicial(bloco64bits) #A permutação inicial é equivalente à

#permutação final inversa.

#Passo 2: Cisão em blocos de 32 bits e 16 iterações

if operacao == 'criptografar':

L = bloco64bits[:32] #Os primeiros 32 bits formam o bloco L.

R = bloco64bits[32:] #Os últimos 32 bits formam o bloco R.

for n in range(1, 17):

L\_n = R #O bloco L da rodada recebe o valor do bloco R anterior.

R\_n = portaXOR(funcao\_f(R, chaveBits, n), L) #O bloco R da rodada recebe o valor da

#operação XOR entre os bits da função f, que usa o bloco R anterior e a sub-chave da

#rodada, e os bits do bloco L anterior.

L = L\_n #Armazenando o valor de L que será utilizado na próxima rodada.

R = R\_n #Armazenando o valor de R que será utilizado na próxima rodada.

bloco64bits = R\_n + L\_n #O L e o R da rodada 16 formam um bloco de 64 bits, com o R

#formando os primeiros bits e o L os últimos.

if operacao == 'descriptografar':

R\_n = bloco64bits[:32] #os primeiros 32 bits formam o bloco R da rodada 16.

L\_n = bloco64bits[32:] #os últimos 32 bits formam o bloco L da rodada 16.

for n in range(16, 0, -1):

R = L\_n #O bloco R da rodada anterior recebe o valor do bloco L da rodada.

L = portaXOR(funcao\_f(L\_n, chaveBits, n), R\_n) #O bloco L da rodada anterior recebe

#o valor da operação XOR entre os bits da função f, que usa o bloco L da rodada e a

#sub-chave da rodada, e os bits do bloco R da rodada.

R\_n = R #armazenando o valor do R que será utilizado na próxima rodada.

L\_n = L #armazenando o valor do L que será utilizado na próxima rodada.

bloco64bits = L + R #Reunindo os valores do primeiro R e do primeiro L formados na

#criptografia para obter o bloco da primeira permutação.

#Passo 3: Permutação final

bloco64bits = permutacaoFinal(bloco64bits) #A permutação final é equivalente à permutação

#inicial inversa.

traducaoBits.extend(bloco64bits) #Adiciona ao vetor criado anteriormente os bits do bloco

#de 64 bits permutado.

return traducaoBits #Vetor com todos os bits da mensagem traduzida.

def gerarSubChave(chave64bits, rodada):

#Recebe o vetor de bits da chave e o número da rodada, retorna a sub-chave da rodada.

#Passo 1: Permutação PC - 1

chave56bits = permutacaoPC1(chave64bits)

#Passo 2: Cisão em blocos de 28 bits e rotação à esquerda

C = chave56bits[:28] #Os 28 primeiros bits formam o bloco C.

D = chave56bits[28:] #Os 28 últimos bits formam o bloco D.

#Neste programa os blocos C e D da rodada são diretamente gerados rotacionando os blocos C e D

#iniciais.

rotacionarEsquerda(C, rodada) #Rotacionando o bloco C.

rotacionarEsquerda(D, rodada) #Rotacionando o bloco D.

#Passo 3: Permutação PC - 2

subChave\_rodada = permutacaoPC2(C + D) #Os blocos C e D rotacionados são agrupados num único

#bloco que será permutado.

return subChave\_rodada

def permutacaoPC1(chave64bits):

#Permutação que remove os bits de paridade da chave e resulta em um bloco de 56 bits.

chave56bits = [] #Vetor sem bits.

for i in permutacao.PC1:

#O looping itera sobre cada número dentro do vetor PC1(variável do módulo permutacao).

chave56bits.append(chave64bits[i]) #O bit do vetor chave64bits no índice i é adicionado ao

#vetor chave56bits.

#O vetor chave56bits contém os bits permutados da chave, note que os bits de paridade foram

#ignorados.

return chave56bits

def permutacaoPC2(chave56bits):

#Permutação que produz as sub-chaves de 48 bits.

chave48bits = [] #Vetor sem bits.

for i in permutacao.PC2:

#O looping itera sobre cada número dentro do vetor PC2(variável do módulo permutacao).

chave48bits.append(chave56bits[i]) #O bit do vetor chave56bits no índice i é adicionado ao

#vetor chave48bits.

#A partir de um vetor de 56 bits se obtém uma sub-chave de 48 bits.

return chave48bits

def permutacaoInicial(bloco64bits):

bloco64bitsPermutado = [] #Vetor sem bits.

for i in permutacao.IP:

#O looping itera sobre cada número dentro do vetor IP(variável do módulo permutacao).

bloco64bitsPermutado.append(bloco64bits[i]) #O bit do vetor bloco64bits no índice i é

#adicionado ao vetor bloco64bitsPermutado.

return bloco64bitsPermutado

def funcao\_f(bloco32bits, chave, rodada):

#Opera com um bloco de 32 bits e uma sub-chave de 48 bits, produzindo um bloco de 32 bits.

subChave = gerarSubChave(chave, rodada) #Gera a sub-chave da rodada.

#Passo 1: Expansão E

E = funcaoExpansao(bloco32bits) #Produz um bloco de 48 bits.

#Passo 2: XOR com a sub-chave da rodada

xor = portaXOR(E, subChave) #XOR com dois blocos de 48 bits.

#Passo 3: Substituição usando caixas-S

bloco32bits = substituicaoCaixas\_S(xor)

#Passo 4: Permutação

f = permutar(bloco32bits)

return f

def funcaoExpansao(bloco32bits):

#Expande um bloco de 32 bits para 48 bits.

bloco48bits = [] #Vetor sem bits.

for i in permutacao.E:

#O looping itera sobre cada número dentro do vetor E(variável do módulo permutacao).

bloco48bits.append(bloco32bits[i]) #O bit do vetor bloco32bits no índice i é adicionado ao

#vetor bloco48bits.

#Note que para aumentar o tamanho do bloco de 32 bits, alguns bits são repetidos durante a

#permutação.

return bloco48bits

def substituicaoCaixas\_S(bloco48bits):

#Separa um bloco de 48 bits em oito blocos de 6 bits que são transformados em oito blocos de 4

#bits e depois os reune para formar um bloco de 32 bits.

#As caixas-S são tabelas com linhas e colunas, neste programa elas estão armazenadas no vetor

#S\_BOX(variável do módulo permutacao) que contém 8 matrizes que representam as tabelas.

bloco32bits = [] #Vetor sem bits.

for caixa\_S in range(8):

#Iteração para separar os oito blocos de 6 bits e as oito caixas-S.

bloco6bits = bloco48bits[caixa\_S \* 6 : (caixa\_S + 1) \* 6] #O número Caixa\_S é utilizado para

#criar o intervalo de 6 bits.

linha = int(bloco6bits[0] + bloco6bits[5], 2) #O primeiro e o último bit do bloco formam um

#número decimal que define a linha.

coluna = int(''.join(bloco6bits[1:5]), 2) #Os quatro bits centrais formam um número decimal

#que define a coluna.

bloco4bits = (bin(permutacao.S\_BOX[caixa\_S][linha][coluna]))[2:] #O número Caixa\_S indica

#qual matriz será usada, o número decimal dessa matriz que estiver na linha e coluna

#indicadas é convertido para binário. Essa conversão resulta em uma string com os dois

#primeiros caracteres indicando que se trata de um número binário, esses caracteres não são

#necessários para o programa e por isso são cortados.

bloco4bits = '0' \* (4 - len(bloco4bits)) + bloco4bits #Se a string tiver menos que 4 dígitos

#binários, adiciona-se zeros à esquerda para que ela tenha exatamente 4 dígitos.

bloco32bits.extend(list(bloco4bits)) #Transforma a string em um vetor de bits e adiciona

#cada bit ao vetor bloco32bits.

return bloco32bits

def permutar(bloco32bits):

bloco32bitsPermutado = [] #Vetor sem bits.

for i in permutacao.P:

#O looping itera sobre cada número dentro do vetor P(variável do módulo permutacao).

bloco32bitsPermutado.append(bloco32bits[i]) #O bit do vetor bloco32bits no índice i é

#adicionado ao vetor bloco32bitsPermutado.

return bloco32bitsPermutado

def permutacaoFinal(bloco64bits):

bloco64bitsPermutado = [] #Vetor sem bits.

for i in permutacao.FP:

#O looping itera sobre cada número dentro do vetor FP(variável do módulo permutacao).

bloco64bitsPermutado.append(bloco64bits[i]) #O bit do vetor bloco64bits no índice i é

#adicionado ao vetor bloco64bitsPermutado.

return bloco64bitsPermutado

#Funções auxiliares do processo de criptografia/descriptografia:

def rotacionarEsquerda(chave28bits, rodada):

#Executa o processo de rotação usado na geração de sub-chaves.

rotacoes = permutacao.ROTACOES[rodada] #Dentro do dicionário do módulo permutacao, o valor

#atribuído à chave de número correspondente ao número da rodada é armazenado. Esse valor é o

#número de rotações a serem realizadas.

for i in range(rotacoes):

bitRemovido = chave28bits.pop(0) #remove o primeiro bit do vetor.

chave28bits.append(bitRemovido) #adiciona o bit removido ao vetor.

#Note que ao retirar o primeiro bit, todos os outros bits se deslocam para a esquerda de

#sua posição original.

#Esta função não necessita de retorno porque ela faz uma cópia da referência da lista passada

#como argumento, ou seja, todas as alterações feitas com a referência da lista consequentemente

#alteram a lista em si.

def portaXOR(A, B):

#Porta lógica XOR bit a bit com dois vetores de bits de mesmo tamanho.

saida = []

for bit in range(len(A)):

#O looping itera sobre o tamanho de um dos vetores.

#A operação lógica é feita entre os bits com o mesmo índice.

if A[bit] != B[bit]:

saida.append('1') #Se os bits forem diferentes a saída é 1.

else:

saida.append('0') #Se os bits forem iguais a saída é 0.

return saida

#Funções de conversão:

def hexaBits(hexadecimal):

#Recebe uma string hexadecimal, cada dois dígitos hexadecimais são convertidos para o byte

#correspondente e retorna um vetor de bits com tamanho divisivel por 64.

vetorBits = [] #Vetor sem bits.

for i in range(0, len(hexadecimal), 2):

#Iteração que ajuda a separar os dígitos hexadecimais em grupos de 2 dígitos.

numeroHex = hexadecimal[i : i + 2] #Seleciona dois dígitos da string.

byte = bin(int(numeroHex, 16))[2:] #Converte o número hexadecimal para decimal e depois para

#binário. Essa conversão resulta em uma string com os dois primeiros caracteres indicando

#que se trata de um número binário, esses caracteres não são necessários para o programa e

#por isso são cortados.

byte = '0' \* (8 - len(byte)) + byte #Se a string tiver menos que 8 dígitos binários,

#adiciona-se zeros à esquerda para que ela tenha exatamente 8 dígitos.

vetorBits.extend(list(byte)) #Transforma a string em um vetor de bits e adiciona cada bit ao

#vetor vetorBits.

resto = len(vetorBits) % 64 #Armazena o resto da divisão do tamanho do vetor de bits por 64.

if resto:

#Se o resto não for 0, adiciona-se zeros ao vetor para que seu tamanho seja divisível por 64.

vetorBits.extend(['0'] \* (64 - resto))

return vetorBits

def bitsHexa(vetorBits):

#Converte um vetor de bits para uma string de dígitos hexadecimais.

hexadecimal = []

for i in range(0, len(vetorBits), 8):

#Iteração que ajuda a separar os bits em grupos de bytes.

byte = ''.join(vetorBits[i : i + 8]) #Seleciona oito bits do vetor e forma uma string.

numeroHex = hex(int(byte, 2))[2:] #Converte os bits para um número decimal e depois para

#hexadecimal. Essa conversão resulta em uma string com os dois primeiros caracteres

#indicando que se trata de um número hexadecimal, esses caracteres não são necessários para

#o programa e por isso são cortados.

numeroHex = '0' \* (2 - len(numeroHex)) + numeroHex #Se a string tiver apenas 1 dígito

#hexadecimal, adiciona-se um zero à esquerda para que ela tenha exatamente 2 dígitos.

hexadecimal.append(numeroHex)

return ''.join(hexadecimal).upper() #Retorna as strings do vetor hexadecimal concatenadas e em

#maiúsculo.

def stringBits(string):

#Converte uma string de caracteres para um vetor de bits.

hexadecimal = []

for char in string:

#Itera sobre todos os caracteres.

numeroHex = hex(ord(char))[2:] #O caractere é convertido para um número decimal da tabela

#ASCII e depois para hexadecimal. Essa conversão resulta em uma string com os dois primeiros

#caracteres indicando que se trata de um número hexadecimal, esses caracteres não são

#necessários para o programa e por isso são cortados.

numeroHex = '0' \* (2 - len(numeroHex)) + numeroHex #Se a string tiver apenas 1 dígito

#hexadecimal, adiciona-se um zero à esquerda para que ela tenha exatamente 2 dígitos.

hexadecimal.append(numeroHex)

vetorBits = hexaBits(''.join(hexadecimal)) #Concatena as strings do vetor hexadecimal e

#converte para bits.

return vetorBits

def bitsString(vetorBits):

#Converte um vetor de bits para uma string de caracteres.

string = []

hexadecimal = bitsHexa(vetorBits) #Converte o vetor para hexadecimal.

for i in range(0, len(hexadecimal), 2):

#Iteração que ajuda a separar os dígitos hexadecimais em grupos de 2 dígitos.

numeroHex = hexadecimal[i : i + 2] #Seleciona dois dígitos da string.

char = chr(int(numeroHex, 16)) #O número hexadecimal é convertido para decimal e depois para

#um caractere da tabela ASCII.

string.append(char)

return ''.join(string) #Retorna os caracteres do vetor concatenados.

if \_\_name\_\_ == '\_\_main\_\_':

main()

#Módulo de variáveis com todas as permutações da criptografia DES

#Os números armazenados servirão de índices para realizar a permutação dos

#vetores no programa principal.

PC1 = (

56, 48, 40, 32, 24, 16, 8,

0, 57, 49, 41, 33, 25, 17,

9, 1, 58, 50, 42, 34, 26,

18, 10, 2, 59, 51, 43, 35,

62, 54, 46, 38, 30, 22, 14,

6, 61, 53, 45, 37, 29, 21,

13, 5, 60, 52, 44, 36, 28,

20, 12, 4, 27, 19, 11, 3

) #Permuted choice 1

PC2 = (

13, 16, 10, 23, 0, 4,

2, 27, 14, 5, 20, 9,

22, 18, 11, 3, 25, 7,

15, 6, 26, 19, 12, 1,

40, 51, 30, 36, 46, 54,

29, 39, 50, 44, 32, 47,

43, 48, 38, 55, 33, 52,

45, 41, 49, 35, 28, 31

) #Permuted choice 2

IP = (

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7,

56, 48, 40, 32, 24, 16, 8, 0,

58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6

) #Initial permutation

E = (

31, 0, 1, 2, 3, 4,

3, 4, 5, 6, 7, 8,

7, 8, 9, 10, 11, 12,

11, 12, 13, 14, 15, 16,

15, 16, 17, 18, 19, 20,

19, 20, 21, 22, 23, 24,

23, 24, 25, 26, 27, 28,

27, 28, 29, 30, 31, 0

) #Expansion function

S\_BOX = (

(

(14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7),

( 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8),

( 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0),

(15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13)

),

(

(15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10),

( 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5),

( 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15),

(13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9)

),

(

(10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8),

(13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1),

(13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7),

( 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12)

),

(

( 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15),

(13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9),

(10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4),

( 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14)

),

(

( 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9),

(14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6),

( 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14),

(11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3)

),

(

(12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11),

(10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8),

( 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6),

( 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13)

),

(

( 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1),

(13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6),

( 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2),

( 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12)

),

(

(13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7),

( 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2),

( 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8),

( 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11)

)

) #Substitution boxes

P = (

15, 6, 19, 20, 28, 11, 27, 16,

0, 14, 22, 25, 4, 17, 30, 9,

1, 7, 23, 13, 31, 26, 2, 8,

18, 12, 29, 5, 21, 10, 3, 24

) #Permutation

FP = (

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25,

32, 0, 40, 8, 48, 16, 56, 24

) #Final Permutation

ROTACOES = {

1:1, 2:2, 3:4, 4:6, 5:8, 6:10, 7:12, 8:14, 9:15,

10:17, 11:19, 12:21, 13:23, 14:25, 15:27, 16:28

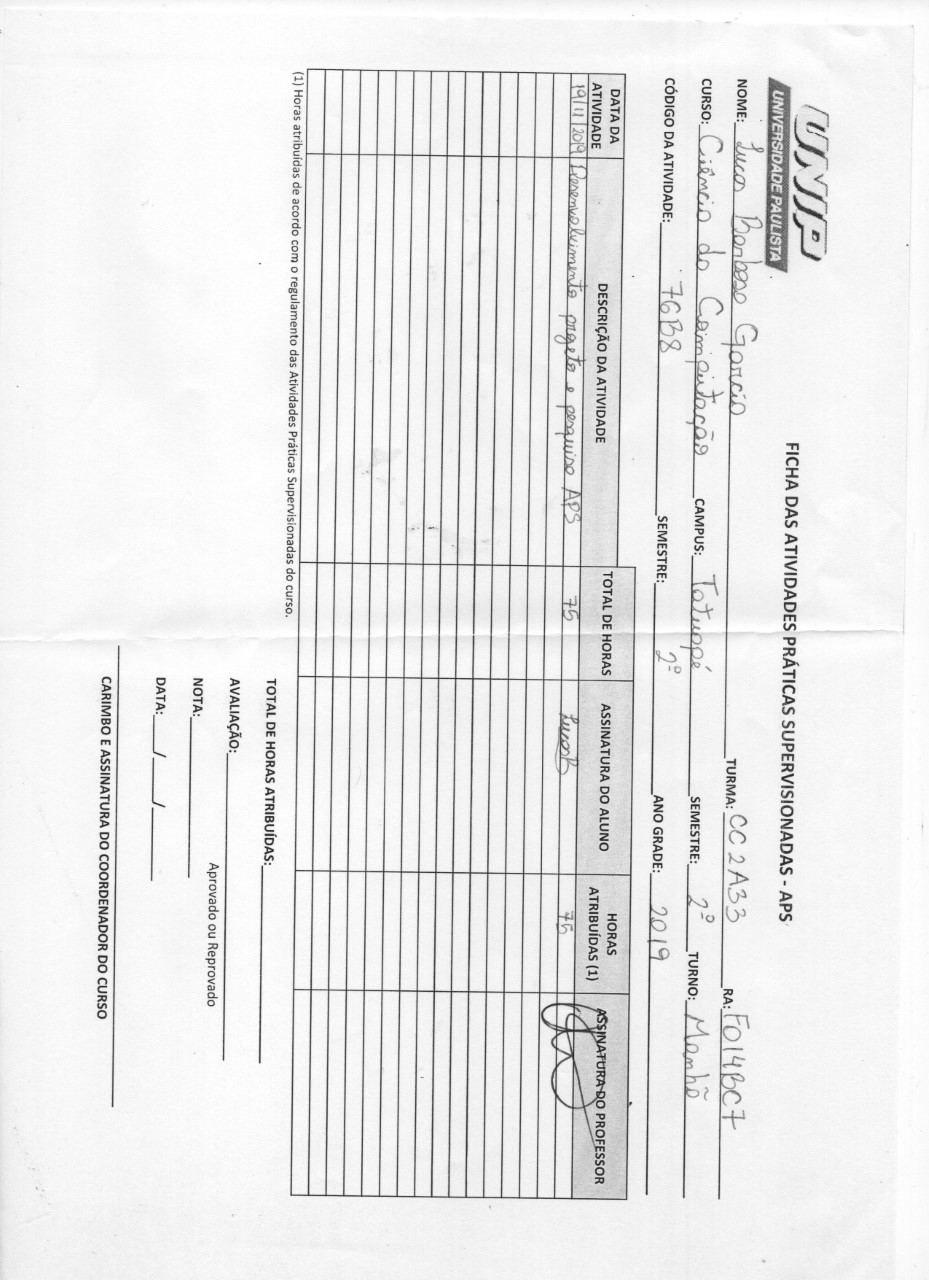
} #Dicionário com chaves que vão de 1 até 16, essas chaves correspondem

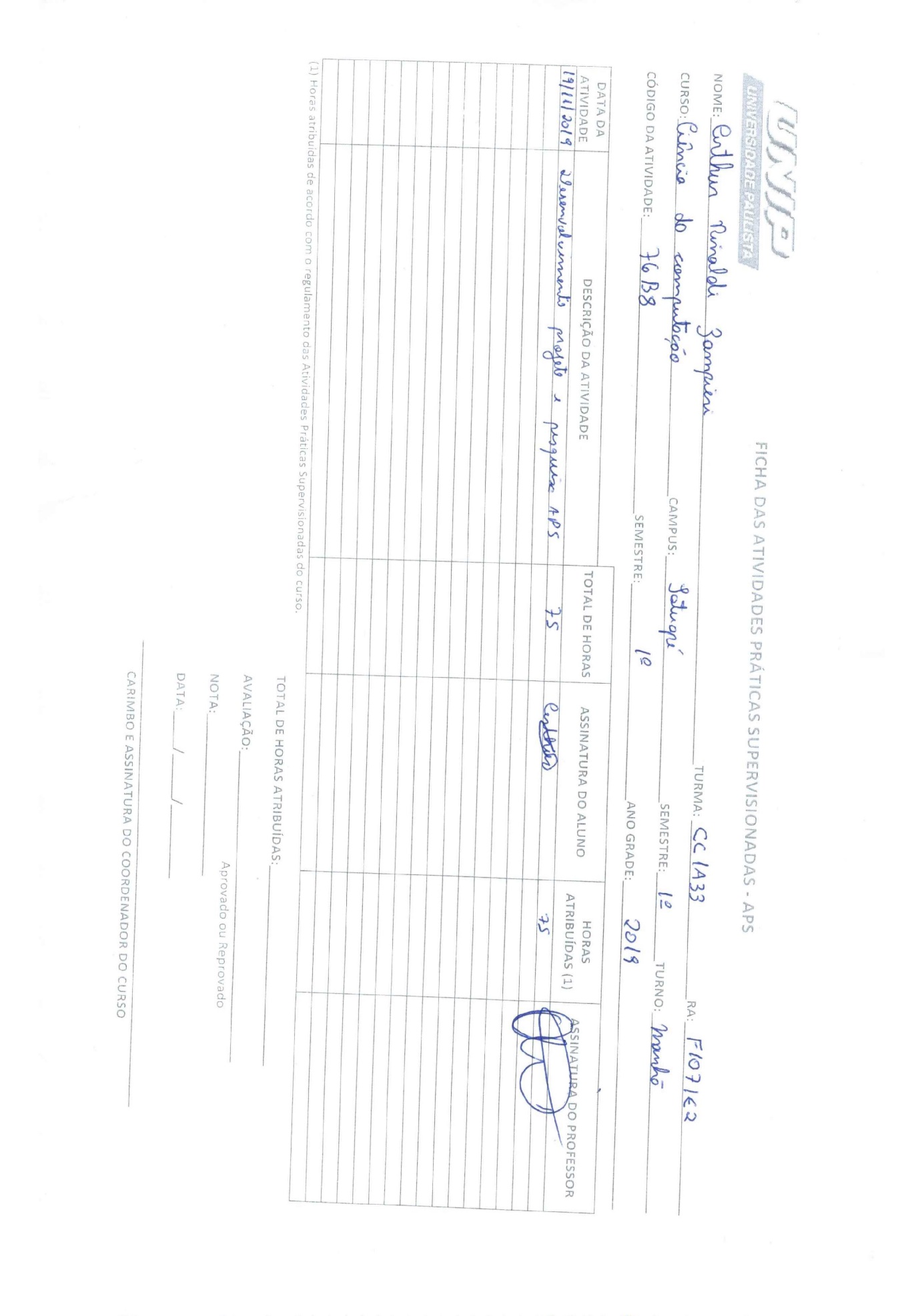
#a cada iteração do processo de criptografia/descriptografia. Os valores

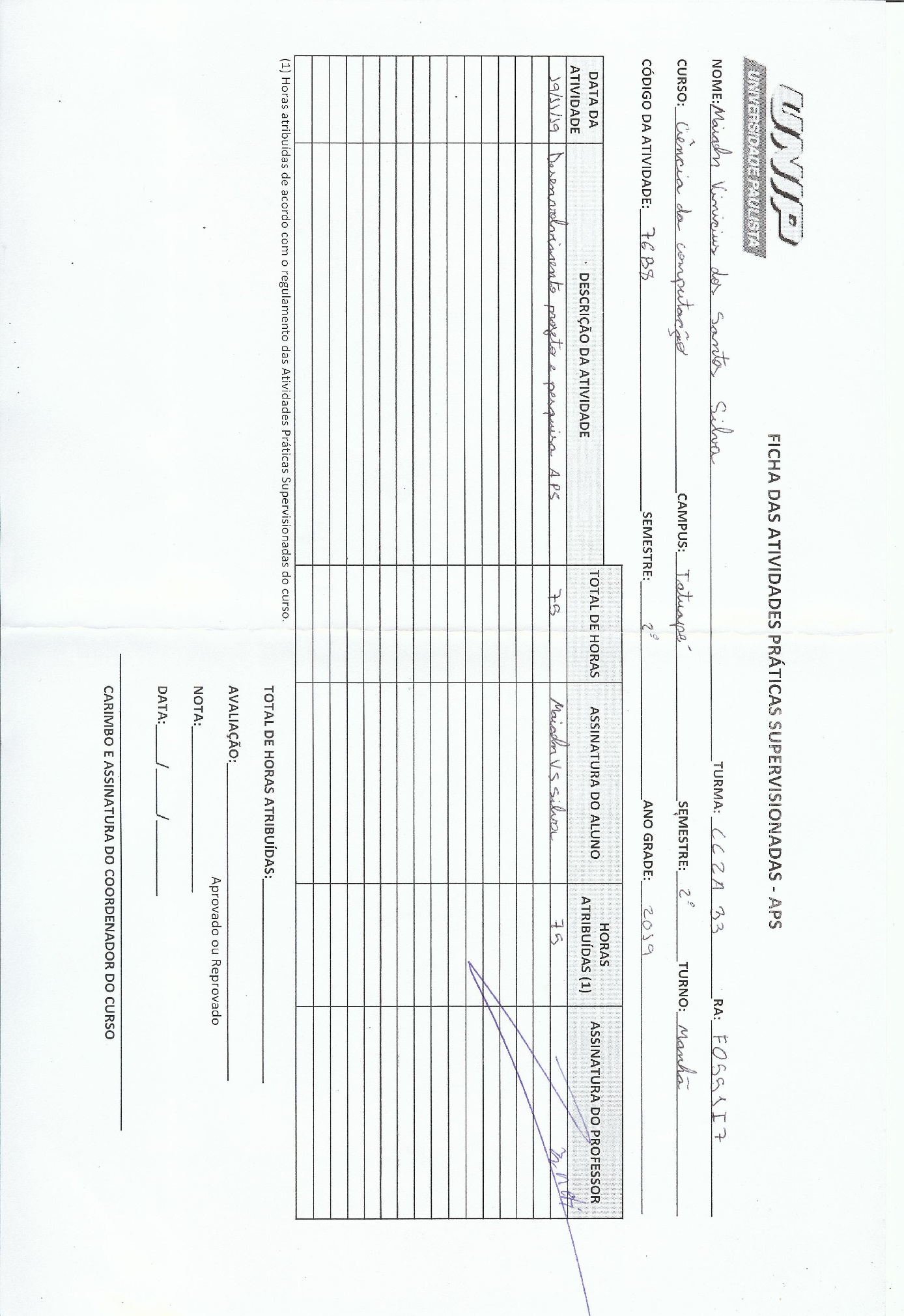
#atribuídos às chaves são o número de rotações necessárias para se obter a

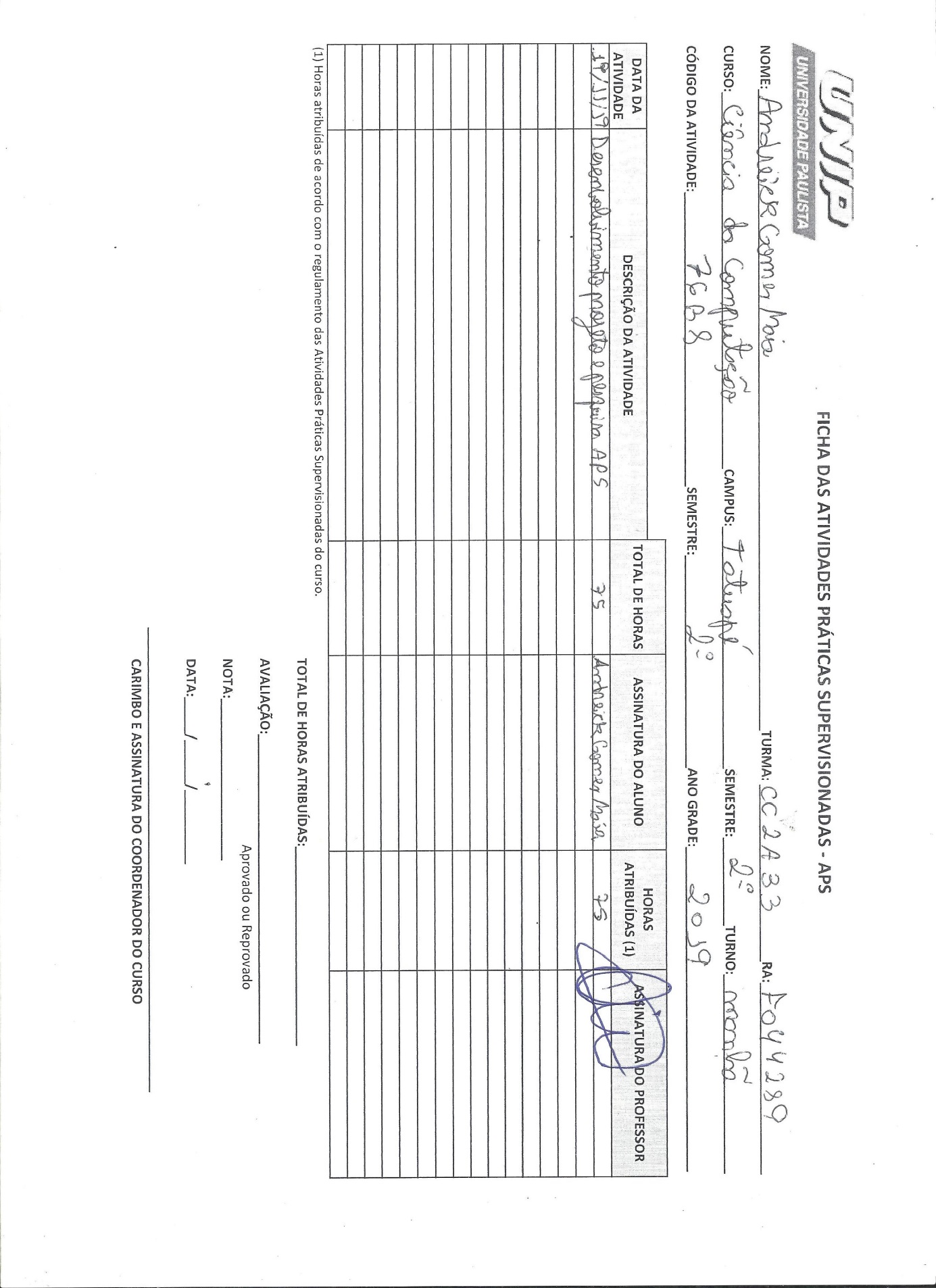
#respectiva sub-chave da iteração.

# FICHAS APS









# CONCLUSÃO

Com base nas pesquisas realizadas para a elaboração do trabalho aprendemos que a criptografia é uma técnica utilizada para codificar mensagens que o homem sempre sentiu necessidade de dissimular, ou seja, de torná-las ininteligíveis, de modo que apenas o emissor e o receptor consigam compreendê-la.

Também foi possível aprender com a pesquisa que a utilização da criptografia tem hoje um interesse ainda maior no campo das comunicações via Internet, pois as informações circulam em infraestruturas das quais não se podem garantir a confiabilidade e a confidencialidade. A criptografia serve não apenas para preservar a confidencialidade dos dados, mas também para garantir a sua integridade e a sua autenticidade. É através da criptografia que suas informações "estarão" seguras.

Outro ponto importante observado foi que a criptografia baseia-se essencialmente na aritmética, ou seja, transforma as letras que compõem a mensagem em uma sucessão de números e, em seguida, faz cálculos com estes números para alterá-los de maneira a que fiquem incompreensíveis. A base da criptografia são as chaves, usadas tanto para criptografar quanto para descriptografar um conteúdo e são conhecidas como simétricas ou assimétricas. Elas são geradas por algoritmos que criam uma sequência específica de caracteres para cada processo e podem ter tamanhos distintos e, quanto maiores, mais seguras elas se tornam. Embora quanto maior o tamanho mais segurança ofereçam, as chaves podem ser quebradas.

Na pesquisa vimos que o DES é considerado uma proteção básica de poucos bits de chaves privadas e simétricas. Por não ser considerado tão seguro, foi criado o 3DES que permitiu aumentar significativamente a segurança, mas com a grande desvantagem de solicitar mais recursos para a codificação e a decodificação. Com a necessidade de mais segurança foi criado o AES, que também é uma codificação de bloco de chave simétrica, e foi introduzido para substituir o DES, já que este usa uma chave de criptografia muito pequena e o algoritmo é bem mais lento.

Ao fim do trabalho, pode-se concluir que a pesquisa realizada ampliou o nosso conhecimento e forneceu informações importantes para entendermos mais a respeito da origem da palavra criptografia, qual seu signifcado, sua importância, tipos e, principalmente, quais os problemas que envolvem este assunto e possíveis soluções.

# Bibliografia

Alecrim, E. (12 de 08 de 2005). *Info Wester*. Fonte: Info Wester: https://www.infowester.com/criptografia.php

CCM - Comunidade online. (22 de Maio de 2017). *Introdução à codificação DES*. Fonte: CCM - Comunidade online: https://br.ccm.net/contents/132-introducao-a-codificacao-des

Certificadora, V. (06 de 02 de 2019). *Cryptoid*. Fonte: Cryptoid: https://cryptoid.com.br/valid/tipos-de-criptografia-conheca-os-10-mais-usados-e-como-funciona-cada-um/

Gadget-info.com. (2019). *Diferença entre DES (Data Encryption Standard) e AES (Advanced Encryption Standard)*. Fonte: Gadget-info.com: https://pt.gadget-info.com/difference-between-des

Garrett, F. (21 de 06 de 2012). *TechTudo.* Fonte: TechTudo: https://www.techtudo.com.br/artigos/noticia/2012/06/o-que-e-criptografia.html

Grabbe, J. O. (2006). *The DES Algorithm Illustrated.* Fonte: Institut für Mathematik: Institut für Mathematik: http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm

Johnson, T. R. (1995). American Cryptology during the Cold War, 1945–1989. In: T. R. Johnson, *American Cryptology during the Cold War, 1945–1989* (pp. 117 - 262). United States. National Security Agency.

Menezes, A. J. (1997). *Handbook of applied cryptography.* Boca Raton : CRC Press.

Morais, J. (08 de 11 de 2017). *Embarcados*. Fonte: Embarcados: https://www.embarcados.com.br/criptografia-aes/

NIST Computer Security Resource Center | CSRC. (19 de Maio de 2005). *FIPS 46-3, Data Encryption Standard (DES).* Fonte: NIST Computer Security Resource Center | CSRC: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf

Nogueira, M. (20 de 08 de 2015). *Estudo Prático*. Fonte: Estudo Prático: https://www.estudopratico.com.br/criptografia/

Nunes, D. S. (04 de 11 de 2007). *PKI – Public key infrastructure*. Fonte: PKI – Public key infrastructure: https://www.gta.ufrj.br/grad/07\_2/delio/Criptografiaassimtrica.html

Pelzl, J., & Paar, C. (26 de Novembro de 2010). *ch03-DES-slides-pt.pdf.* Fonte: INSTITUTO DE COMPUTAÇÃO UNICAMP: http://www.ic.unicamp.br/~rdahab/cursos/mo421-mc889/2014-1s/Welcome\_files/SlidesPaarPelzl-Portugues-pdfs/ch03-DES-slides-pt.pdf

Pelzl, J., & Paar, C. (2010). *Entendendo Criptografia – Um Livro Texto para Estudantes e Profissionais.* Heidelberg, Dordrecht, London, New York: Springer.

Rafael Santos Andrade, F. d. (2012). ALGORITMO DE CRIPTOGRAFIA RSA. *Eventos Pedagógicos*, 443-444.

Romagnolo, C. A. (24 de 08 de 2007). Fonte: Oficina da net: https://www.oficinadanet.com.br/artigo/443/o\_que\_e\_criptografia

Schneier, B. (6 de Outubro de 2004). *The Legacy of DES - Schneier on Security*. Fonte: Schneier on Security: https://www.schneier.com/blog/archives/2004/10/the\_legacy\_of\_d.html

*Segurança da Informação*. (s.d.). Fonte: Segurança da Informação: http://seguranca-da-informacao.info/criptografia.html

Soares, J. (22 de 02 de 2017). *codigosimples.net*. Fonte: CodigoSimpes: https://codigosimples.net/2017/02/22/introducao-aos-algoritmos-de-criptografia-aes-e-des-com-net/