DESENVOLVIMENTO DE UMA FERRAMENTA DE CRIPTOGRAFIA DE DADOS PRODUÇÃO DE UMA FERRRAMENTA DE CRIPTOGRAFIA E DESCRIPTOGRAFIA PARA PROTEÇÃO DE DADOS ATRAVÉS DE UMA INTERFACE GRÁFICA

Cap **Flávio** Barros Correia
Cap Diego Madureira **Peixoto**
Cap Cassius Matheus Alves **Bierhals**

RESUMO

Este estudo tem por finalidade fornecer informações que possam contribuir para o desenvolvimento de uma ferramenta de criptografia e descriptografia de dados que atendam às demandas das operações militares conduzidas pelo Exército Brasileiro. A partir de uma análise sumária, da literatura existente sobre o tema, serão definidos os requisitos para o desenvolvimento de uma ferramenta que implementa algoritmos de criptografia e descriptografia em uma interface gráfica, garantindo segurança, desempenho e simplicidade para que o usuário final possa proteger suas informações das diversas ameaças cibernéticas atuais, de forma eficiente e segura.

Palavras-chave: criptografia, algoritmo, dados, segurança, exército

1 INTRODUÇÃO

A finalidade deste trabalho é apresentar como produto final uma aplicação digital que seja capaz de criptografar e descriptografar dados com uma interface gráfica intuitiva que atenda às necessidades de uma operação militar do Exército Brasileiro.

1.1 CONTEXTUALIZAÇÃO DO ESTUDO

Criptografar uma informação consiste em torná-la ininteligível para quem não deveria possuir acesso a ela. É essencial para qualquer instituição que precise manter sigilo sobre seus dados ou protegê-los de potenciais ameaças cibernéticas.

Com o avanço da tecnologia e o advento da Guerra Cibernética, torna-se fundamental a criação de medidas de proteção que garantam a segurança da informação no âmbito do Exército Brasileiro, sobretudo nas operações militares.

Os desafios de gerenciar riscos, evitar ameaças e mitigar danos estão essencialmente conectados ao estabelecimento de uma rede de comunicações segura e eficiente.

Diante desta realidade, faz-se necessário o fortalecimento da mentalidade de segurança da informação. O desenvolvimento de uma ferramenta de criptografia de dados confiável e de fácil operação é uma maneira concreta de contribuir para esta mentalidade.

1.2 JUSTIFICATIVA

O Exército Brasileiro, enquanto uma das instituições responsáveis pela defesa da soberania nacional, deve estar constantemente capacitado para garantir que seus dados permaneçam inacessíveis para usuários não autorizados.

Uma falha de segurança que prejudique o princípio da confidencialidade das informações pode colocar em circulação inapropriada desde detalhes operacionais e logísticos até dados sensíveis sobre a tropa.

Assegurar a proteção de dados é um fator determinante, tanto para a manutenção da imagem da Força, quanto para a segurança e a eficácia de uma operação militar.

1.3 DEFINIÇÃO DO PROBLEMA DE PESQUISA

A evolução dos meios tecnológicos aumenta a demanda por sistemas de segurança e exige que as ferramentas existentes de segurança da informação estejam constantemente se atualizando.

Paralelo a este fator, faz-se necessária a

adaptação desses recursos às necessidades das operações militares do Exército Brasileiro, de modo que possam ser empregados, inclusive, por indivíduos não qualificados, através de uma aplicação que indique intuitivamente as etapas de um processo de criptografia ou descriptografia de uma informação.

1.4 OBJETIVOS DA PESQUISA

Esta pesquisa busca contribuir para o desenvolvimento de uma ferramenta de criptografia que atenda às demandas de proteção de dados por parte do Exército Brasileiro contra potenciais ameaças internas e externas em operações militares.

1.5 ESTRUTURA DO CONTEÚDO ESCRITO

O artigo em sua primeira seção contextualiza a importância da criptografia no campo da proteção cibernética, abordando os princípios da segurança da informação.

A Revisão de Literatura apresenta conceitos importantes para o estudo tais como os de criptografia simétrica e assimétrica, alguns dos principais algoritmos de criptografia, - AES (Advanced Encryption Standard) e RSA (Rivest-Shamir-Adleman) - oferecendo uma análise comparativa de sua eficiência e segurança, e o conceito de *hash* Serão incluídas referências a autores conceituados sobre segurança em redes de computadores e criptografia, que exploram essas ideias.

Na seção de Métodos de Pesquisa, será apresentada a concepção geral da ferramenta a ser desenvolvida. Serão identificadas as funcionalidades essenciais que a aplicação deve possuir, as etapas do processo de criptografia e descriptografia e a criação de uma interface gráfica intuitiva. Um exemplo prático seria a inclusão de uma funcionalidade para criptografar e descriptografar arquivos como documentos PDF, diretamente pela interface. Os dados coletados para avaliação da eficácia da ferramenta serão obtidos por meio de testes da mesma por potenciais usuários.

A seção de Implementação abordará as tecnologias utilizadas, como *Python* para a programação da ferramenta e o uso de bibliotecas como *PyCryptodome* para a implementação dos algoritmos de criptografia. Já a seção sobre a Interface Gráfica explicará a criação de um design intuitivo utilizando *frameworks* como *Tkinter*.

Na parte de Testes de Segurança e Desempenho, serão detalhados os testes realizados para avaliar a robustez da ferramenta contra-ataques de força bruta e o tempo de processamento em diferentes cenários.

Por fim, a Documentação e POP (Procedimentos Operacionais Padrão) apresentarão as instruções de uso e manutenção da ferramenta, acompanhada de exemplos práticos de utilização em operações militares.

2 DESENVOLVIMENTO

2.1 REVISÃO DA LITERATURA

A criptografia é uma forma de segurança que permite a troca segura de informações em um mundo ameaçado. Para Schneier (1996, p. 21), “é a arte e a ciência de manter mensagens seguras” (tradução nossa).

O processo de criptografar uma informação consiste em torná-la inacessível para um indivíduo não autorizado. É uma prática que busca manter a confidencialidade, a autenticidade e a integridade de um dado.

A criptografia é essencial para proteger dados sensíveis que circulam em redes de comunicação, especialmente em situações de combate ou operações de inteligência. Não por acaso, como explica Tanenbaum (2011, p. 148), os militares tiveram papel importante no desenvolvimento dessa arte e definiram as bases para futuras tecnologias.

2.1.1 CRIPTOGRAFIA SIMÉTRICA

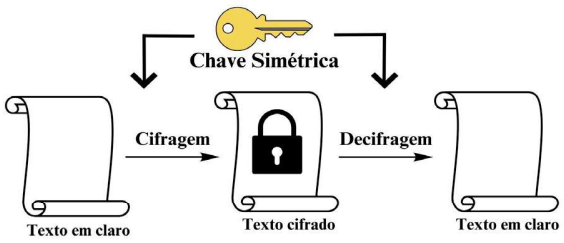
Explica Stallings (2015, p. 21), que a criptografia simétrica ou de chave privada, é uma técnica em que a mesma chave é utilizada para cifrar e decifrar dados, exigindo que ambas as partes da comunicação compartilhem essa chave de forma segura.

A utilização de uma chave criptográfica única permite a implementação deste método de segurança de maneira facilitada e é preferível para situações que priorizem a velocidade e a maior quantidade de

informação, por conta do menor consumo de recursos dos equipamentos disponíveis.

Ferguson, Schneier e Kohno concordam que (2010, p. 28) a criptografia simétrica oferece maior eficiência computacional, sendo adequada para sistemas onde há grande volume de dados a serem processados em tempo real.

FIGURA 1 - Criptografia Simétrica.



Fonte: os autores.

2.1.2 AES (*ADVANCED ENCRYPTION STANDARD*)

O algoritmo AES foi desenvolvido como um sucessor ao algoritmo DES (*Data Encryption Standard*), que se tornou vulnerável a ataques de força bruta devido ao aumento das capacidades operacionais dos equipamentos modernos.

Diante deste novo cenário, o AES foi projetado para ser rápido, seguro e eficiente, com a capacidade de operar com chaves de diferentes tamanhos, o que aumenta sua segurança contra-ataques.

Buscando transpor este obstáculo da vulnerabilidade a ataques de força bruta, o algoritmo AES foi estruturado para operar com três tamanhos de chave: 128, 192 e 256 bits, enquanto o anterior, DES, destinava 56 bits da chave para a cifração da informação.

A principal vantagem do AES é a sua robustez contra-ataques de força bruta, sendo que, com chaves de 256 bits, o número de possíveis combinações torna praticamente inviável qualquer tentativa de quebra do algoritmo com a tecnologia atual.

O algoritmo AES, desta maneira, conforme Daemen e Rijmen (2002, p. 147) permite uma grande flexibilidade no comprimento do bloco sem perder as propriedades de eficiência e alta resistência contra criptoanálise. Este equilíbrio entre segurança e desempenho, permite sua aplicação em várias plataformas e protocolos, desde dispositivos móveis até redes de alto desempenho.

TABELA 1 - Estimativas para ataques de 'força bruta' em algoritmos simétricos.

Custo	56 bits	64 bits	112 bits	128 bits
$100 K	3,5 horas	37 dias	10^13 anos	10^18 anos
$1 M	21 minutos	4 dias	10^12 anos	10^17 anos
$10 M	2 minutos	9 horas	10^11 anos	10^16 anos
$100 M	13 segundos	1 hora	10^10 anos	10^15 anos
$1 G	1 segundo	5,4 minutos	10^9 anos	10^14 anos
$10 G	0,1 segundos	32 segundos	10^8 anos	10^13 anos
$100 G	0,01 segundos	3 segundos	10^7 anos	10^12 anos
$1 T	1 milissegundo	0,3 segundos	10^6 anos	10^11 anos

Fonte: NAKAMURA, Emílio Tissato. GEUS, Paulo Lício de. *Segurança de Redes em Ambientes Corporativos*. 1ª. ed. São Paulo: Novatec Editora, 2007. Reimpro. 2012, Tabela 9.3, p. 311. Estimativas para ataques de 'força bruta' em algoritmos simétricos.

2.1.3 CRIPTOGRAFIA ASSIMÉTRICA

Stallings (2015, p. 200) afirma que a criptografia assimétrica ou de chave pública oferece uma mudança radical no processo anterior, já que, ao contrário da criptografia simétrica, não requer que as partes envolvidas compartilhem uma chave secreta.

Esse método de proteção de dados faz uso de um par de chaves nas suas operações: uma chave privada e uma chave pública. Esta pode ser compartilhada abertamente e utilizada para criptografar os dados que se deseja transmitir por um canal de comunicação seguro; aquela deve ser mantida com seu proprietário e utilizada para decifrar os dados recebidos.

"A principal vantagem de tais sistemas é que fornecer chaves públicas autênticas é geralmente mais fácil do que distribuir chaves secretas de uma forma segura, conforme exigido em sistemas de chaves simétricas" (MENEZES, VAN OORSCHOT e VANSTONE, 1996 , p. 283, tradução nossa).

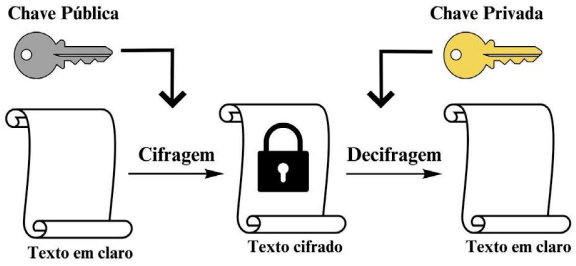
Sua escolha está relacionada às operações que não demandam grande volume de dados ou velocidade, uma vez que consome mais recursos computacionais.

Outra vantagem dessa forma de criptografia é a garantia do princípio de autenticidade da informação. Além de ser necessário o par de chaves do destinatário para que o conteúdo seja devidamente cifrado e decifrado, o par de chaves do remetente

pode ser utilizado para assinar digitalmente a mensagem com a sua chave privada. Dessa maneira, o destinatário, de posse da chave pública do remetente, terá condições de verificar se ela foi enviada por ele.

"As assinaturas digitais permitem um método de assegurar que a mensagem é autêntica para um usuário e que ela de fato se origina da pessoa que alega tê-la enviado" (PAAR e PELZL, 2010, p. 259, tradução nossa). Schneier (1996, p. 62) as compara com as assinaturas manuscritas, como prova de autenticidade.

FIGURA 2 - Criptografia assimétrica.



Fonte: os autores.

2.1.4 RSA (RIVEST-SHAMIR-ADLEMAN)

O algoritmo RSA, Rivest-Shamir-Adleman, se baseia na dificuldade de fatoração de números inteiros grandes, razão pela qual, segundo Stallings (2015, p. 207), desde o seu desenvolvimento segue como a técnica de uso geral mais aceita e implementada para a encriptação de chave pública.

O processo criptográfico do algoritmo se inicia com a seleção de dois números primos grandes, com centenas de dígitos. O produto da multiplicação entre esses dois números gera um valor que será usado nas operações matemáticas seguintes para criar o par de chaves, pública e privada, garantindo a segurança e a integridade da comunicação.

Pela própria natureza matemática do problema, o tempo exponencial para resolvê-lo torna a operação em questão impraticável, mesmo para os computadores mais avançados tecnologicamente.

"Fazendo com que cada um dos fatores tenha 100 dígitos, a multiplicação pode ser feita em uma fração de segundo, mas a fatoração exigiria bilhões de anos, usando o melhor algoritmo conhecido" (HELLMAN, 1978, p. 45, tradução nossa).

Em concordância com Katz e Lindell (2007, p. 231), que relacionam a criptografia moderna a problemas matemáticos, o RSA exemplifica a eficácia da teoria dos números aplicada ao contexto de segurança da informação.

"Ainda que os algoritmos de fatoração estejam constantemente se desenvolvendo, a situação atual ainda está longe de representar uma ameaça para a segurança do RSA, quando ela é usada adequadamente" (BONEH, 1999, p. 204, tradução nossa).

TABELA 2 - Fatoração de chaves do algoritmo assimétrico.

Nº bits	MIPS/Anos necessários	Tempo p/ Pentuim II – 300 MHz
512	< 200	8 meses
728	100.000	300 anos
1024	3 x 107	105 anos
1280	3 x 109	107 anos
1536	2 x 1011	108 anos
2048	4 x 1014	1,3 x 1012 anos

Fonte: NAKAMURA, Emílio Tissato. GEUS, Paulo Lício de. *Segurança de Redes em Ambientes Corporativos*. 1ª. ed. São Paulo: Novatec Editora, 2007. Reimpr. 2012, Tabela 9.4, p. 311. Fatoração de chaves do algoritmo assimétrico.

2.1.5 HASHING

A técnica de *Hashing,* ou geração de *hash*, consiste em transformar, com processos matemáticos, dados de tamanho variável em uma saída de tamanho fixo, a qual se dá o nome de *hash.* Segundo Rivest (1992, p. 5), as funções *hash* (ou *digest*) são utilizadas para criar "impressões digitais" de dados, permitindo que grandes quantidades de informação sejam resumidas em um pequeno valor fixo.

O *hash* gerado é composto de uma sequência de caracteres única, resultado de uma operação unilateral. Isso significa que sua reversão é altamente improvável, uma vez que deveriam ser testadas todas as combinações possíveis para obter a entrada original.

Pelo mesmo motivo de que a quantidade de combinações possíveis para gerar um *hash* dificulta sua reversão, também torna bastante improvável que duas entradas diferentes gerem a mesma saída, ou seja, uma colisão. Apesar de ser possível, os algoritmos modernos são projetados para serem

resistentes a essa remota possibilidade, como por exemplo o algoritmo SHA-256.

"Um hash é um exemplo do que é chamado de função unidirecional, uma função fácil de calcular, mas difícil de inverter, para que a mensagem original não possa ser recuperada" (DIFFIE e LANDAU, 2007, p. 253, tradução nossa).

Essa técnica garante ainda que com a menor alteração nos dados, o *hash* seja alterado completamente. Essa é mais uma maneira de verificar a integridade da informação.

TABELA 3 - O espaço das chaves e o tempo de processamento necessário.

Combinações permitidas	7 bBytes	7 bBytes	8 bBytes	8 bBytes
Letras minúsculas (26)	8 x 10^9	2,2 horas	2,1 x 10^11	2,4 dias
Minúsculas e dígitos (36)	7,8 x 10^10	22 horas	2,8 x 10^12	33 dias
Alfanuméricos (62)	3,5 x 10^12	41 dias	2,2 x 10^14	6,9 anos
Caracteres imprimíveis (95)	7 x 10^13	2,2 anos	6,6 x 10^15	210 anos
Caracteres ASCII (128)	5,6 x 10^14	18 anos	7,2 x 10^16	2300 anos
Caracteres ASCII de 8 bits (256)	7,2 x 10^16	2300 anos	1,8 x 10^19	580000 anos

Fonte: NAKAMURA, Emílio Tissato. GEUS, Paulo Lício de. *Segurança de Redes em Ambientes Corporativos*. 1ª. ed. São Paulo: Novatec Editora, 2007. Reimpr. 2012, Tabela 9.2, p. 310. Fatoração de chaves do algoritmo assimétrico.

2.1.6 PROTEÇÃO DAS COMUNICAÇÕES MILITARES

O desafio da implementação de um sistema de criptografia adequado também é preocupação da Força Terrestre. É um requisito básico para que as operações militares ocorram sem o risco de comprometimento.

A importância da proteção cibernética se dá desde os escalões mais altos, com, por exemplo, a definição de uma Doutrina Militar de Defesa Cibernética (MD31-M-07, 2023), até os níveis mais elementares.

Conforme a Diretriz Estratégica Organizadora do Sistema de Comando e Controle do Exército (EB10-D-01.013, 2021), "a criptografia deve ser empregada de forma extensiva para garantir a segurança da informação e a proteção dos dados críticos em redes de comando e controle" (p. 22).

A segurança cibernética também é abordada no Manual de Guerra Cibernética (EB70-MC-10.232, 2017), que destaca a necessidade de uma infraestrutura cibernética resiliente capaz de suportar ataques e manter a continuidade das operações de comando e controle. O uso do algoritmo AES no Sistema de Comando e Controle da Força Terrestre e em outros sistemas de comunicação demonstra a adaptação do Exército Brasileiro às melhores práticas globais de segurança da informação.

O crescimento das ameaças cibernéticas e dos ataques direcionados a infraestruturas críticas incentiva as Forças Armadas a adotarem uma abordagem proativa no que tange à proteção cibernética.

2.2 MÉTODOS DE PESQUISA

2.2.1 ABORDAGEM

A metodologia seguiu uma abordagem empírica e exploratória, visando implementar soluções robustas de criptografia de dados que garantem a confidencialidade, integridade e autenticidade das informações em cenários críticos. A pesquisa combinou uma revisão bibliográfica com desenvolvimento prático, permitindo identificar e aplicar melhores práticas em segurança cibernética.

2.2.2 DESENVOLVIMENTO DA FERRAMENTA

Através da aplicação *Visual Studio Code*, que permite a edição de códigos de programação, foi desenvolvido em linguagem *Python* um executável que permita a criptografia ou descriptografia baseada em uma combinação dos dois algoritmos referenciados.

O algoritmo AES-256 é utilizado para garantir a confidencialidade e a integridade dos dados transmitidos, enquanto o algoritmo RSA-4096 é empregado no gerenciamento das chaves simétricas geradas. Esta conjunção proporciona o equilíbrio ideal entre segurança e eficiência para a criação de um método robusto de segurança de dados sensíveis.

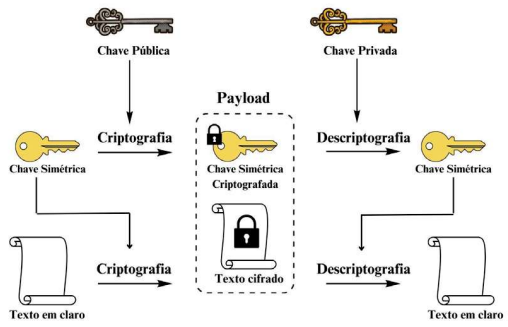
Além dos algoritmos principais, a pesquisa também investigou soluções

complementares, como derivação de chaves seguras e assinaturas digitais. Essas técnicas adicionaram camadas extras de proteção e garantiram a autenticidade das informações, evitando modificações maliciosas durante a transmissão.

O desenvolvimento da solução seguiu um processo gradual, com ajustes realizados conforme necessário ao longo da implementação. Essa abordagem possibilitou uma integração eficiente entre teoria e prática, assegurando que os algoritmos selecionados fossem adequados e atendessem aos requisitos do estudo. A criptografia simétrica (AES-256) foi utilizada para lidar com grandes volumes de dados, enquanto a criptografia assimétrica (RSA-4096) garantiu a proteção das chaves simétricas.

Por fim, a inclusão de assinaturas digitais assegurou a integridade e autenticidade das informações trocadas, resultando em uma solução equilibrada entre segurança e eficiência. A metodologia aplicada garantiu que a solução atendesse às necessidades de segurança em ambientes críticos, como centros de controle e infraestruturas sensíveis.

FIGURA 3 - Criptografia híbrida.



Fonte: os autores.

2.3 APRESENTAÇÃO E ANÁLISE DE DADOS

2.3.1 IMPORTAÇÃO DAS BIBLIOTECAS

A primeira etapa no desenvolvimento do *script* é a importação de bibliotecas necessárias para garantir que todas as funcionalidades da ferramenta de criptografia estejam disponíveis. Cada biblioteca desempenha um papel essencial para o funcionamento adequado da ferramenta, conforme a Tabela 1.

TABELA 4 - Bibliotecas

BIBLIOTECA	FINALIDADE
Tkinter	Criar uma interface gráfica que facilita a interação com o usuário, através de elementos como janelas, botões e caixas de diálogo
PyCryptodome	Fornecer os algoritmos de criptografia e *hash* utilizados
Base64	Converter dados binários (como o arquivo criptografado) para um formato de texto legível
OS	Permitir a manipulação de arquivos e caminhos de arquivos
Datetime	Registrar a data e a hora em que as operações são realizadas, para fins de auditoria e rastreamento

Fonte: os autores

2.3.1 FUNÇÕES

As funções são os elementos que conduzem a operação do *script*. A exposição procurou seguir uma sequência lógica que acompanhe o funcionamento do programa. Conforme a próxima função é acionada, o trabalho fornece sua explicação respectiva.

O código completo e as instruções de uso da aplicação estão disponíveis no Apêndice A deste estudo.

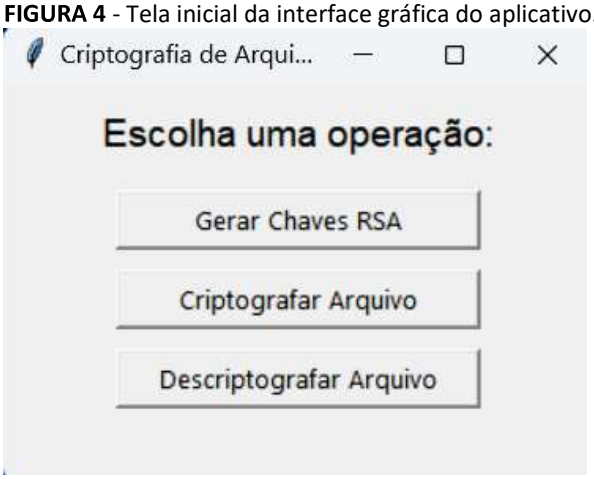
2.3.1.1 INÍCIO DO SCRIPT E CRIAÇÃO DA INTERFACE GRÁFICA

O programa deve ser executado diretamente pelo usuário. Caso não haja erros na execução, a interface gráfica criará uma janela para interação com o usuário.

As opções são apresentadas de forma simples e intuitiva para fácil entendimento do usuário. Cada um dos botões é responsável por

acionar uma função específica: gerar chaves RSA, criptografar e descriptografar arquivos.

FIGURA 4 - Tela inicial da interface gráfica do aplicativo.



Fonte: os autores.

2.3.1.2 GERAR CHAVES RSA

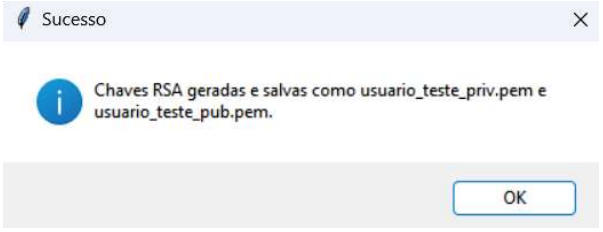
A função do primeiro botão é a geração de um par de chaves utilizando o algoritmo RSA, de criptografia assimétrica.

O programa solicitará um nome a ser usado para salvar os arquivos que contém a chave privada e a chave pública. A chave privada, por sua vez, deverá ser protegida por meio de uma senha também fornecida e confirmada pelo usuário.

Ambas as chaves são geradas por meio do algoritmo RSA com 4096 bits de comprimento. A chave privada será protegida pelo algoritmo AES com 128 bits, fornecendo mais uma camada de segurança.

Ao final da execução, o programa exibe uma mensagem de confirmação do usuário e o par de chaves é salvo no mesmo diretório do *script*.

FIGURA 4 - Mensagem de confirmação do aplicativo.



Fonte: os autores.

2.3.1.3 CRIPTOGRAFAR ARQUIVO

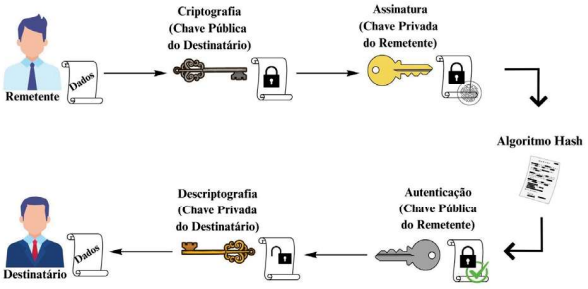
O segundo botão ativa a função para encriptar o arquivo a ser selecionado pelo usuário. Para esta operação é necessário que as chaves sejam geradas na etapa anterior.

O programa, com o algoritmo AES-256, gera uma nova chave simétrica, que é usada para cifrar o arquivo selecionado. Foi inserido como recurso adicional o modo GCM (*Galois/Counter Mode*), que cria uma espécie de *tag* de autenticação que detecta qualquer modificação no arquivo após a criptografia. Se for detectada alguma alteração, a etapa da descriptografia não poderá ser realizada corretamente.

Conforme o funcionamento da criptografia híbrida, a chave pública do destinatário é utilizada para cifrar a segunda chave simétrica, que foi criada nessa mesma etapa, enquanto a chave privada do remetente funciona como assinatura digital, para certificar que o arquivo foi enviado pelo usuário correto.

O arquivo final criptografado é composto pelos seguintes elementos: o próprio conteúdo do arquivo (cifrado com a chave AES-256), a chave AES-256 (cifrada pela chave pública do destinatário), a *tag* de autenticação (gerada pelo modo de operação GCM) e a assinatura digital. O produto é salvo no mesmo diretório com a extensão '.enc' e o programa exibe ao usuário uma mensagem de confirmação.

FIGURA 5 - Processo de criptografia assimétrica com assinatura digital e autenticação com as chaves pública e privada.



Fonte: os autores.

2.3.1.4 DESCRIPTOGRAFAR ARQUIVO

A terceira e última alternativa que a aplicação disponibiliza opera a decifração de um arquivo previamente criptografado. Semelhante à etapa anterior, é necessário importar a chave privada do destinatário e a

chave pública do remetente junto com o arquivo a ser decifrado.

A ferramenta, de posse da chave privada do destinatário, descriptografa o conteúdo do arquivo. Durante esse processo, a *tag* de autenticação é verificada: se houver qualquer alteração no arquivo, o processo falha; caso contrário, segue normalmente.

Com a chave pública do remetente, a assinatura digital é confirmada e por fim, o conteúdo é salvo no formato original e uma mensagem de sucesso é exibida para o usuário.

2.3.1.5 REGISTRO DE *LOGS*

A fim de possibilitar a auditoria ou rastreamento das atividades realizadas pela ferramenta, foi inserida também uma função que armazena todos os registros em um arquivo nomeado “security_log.txt”.

2.4 DISCUSSÃO DOS RESULTADOS

Os testes da ferramenta evidenciaram uma boa aceitação das suas funcionalidades. A interface possibilitou rápida e fácil comunicação com o usuário e os requisitos de segurança foram atendidos,

O programa foi disponibilizado para usuários de um grupo de controle específico que a empregaram para cifrar arquivos de variados formatos e tamanhos.

A aliança dos dois tipos de criptografia, simétrica e assimétrica, permitiu a robustez da codificação e a eficiência no processamento dos dados, resultando em uma aplicação que pode ser aplicada em diferentes cenários operacionais.

3. CONCLUSÃO

Este *script* foi projetado para ser uma ferramenta prática e segura para criptografia de arquivos, utilizando algoritmos robustos como AES e RSA. Cada etapa do processo é cuidadosamente organizada para garantir que os dados estejam protegidos durante todo o ciclo de vida, desde a geração das chaves até a criptografia e descriptografia dos arquivos.

3.1 RESULTADOS

A interface gráfica simples permite que qualquer usuário, independentemente do nível de conhecimento técnico, possa usar a ferramenta com facilidade.

O desenvolvimento deste *script* foi motivado pela necessidade de criar uma solução que possa atender aos desafios de Segurança Cibernética enfrentados em ambientes sensíveis, como Centros de Comando e Controle do Exército Brasileiro ou demais órgãos de Comunicações.

A principal preocupação é a manutenção da confidencialidade, da integridade e da autenticidade dos dados, evitando que informações sensíveis sejam acessadas ou modificadas por agentes não autorizados.

Nesse contexto, a ferramenta de criptografia desenvolvida tem como objetivo garantir a segurança das comunicações e do armazenamento de dados de forma eficiente e prática, mesmo para usuários que não possuem conhecimento técnico avançado.

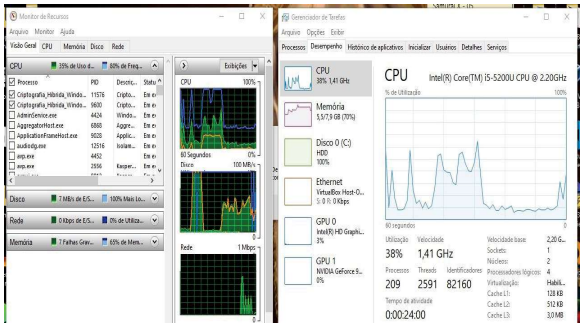
3.2 DESEMPENHO

Com a finalidade de avaliar o desempenho da ferramenta desenvolvida, foi utilizado um computador com as seguintes características: Sistema Operacional Windows 64, CPU i5 (5ª geração) e 8GB de memória RAM, com os resultados conforme tabela e figuras abaixo:

TABELA 5 – Primeira rodada de testes

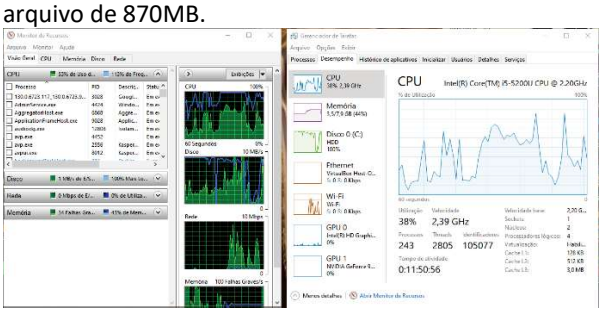
Tamanho do arquivo a ser cifrado	Tempo para cifrar	Tempo para decifrar
175MB	8 segundos	5 segundos
870 MB	50 segundos	30 segundos

FIGURA 5- Processamento durante a criptografia do arquivo de 870MB.



Fonte: os autores

FIGURA 6- Processamento durante a descriptografia do arquivo de 870MB.



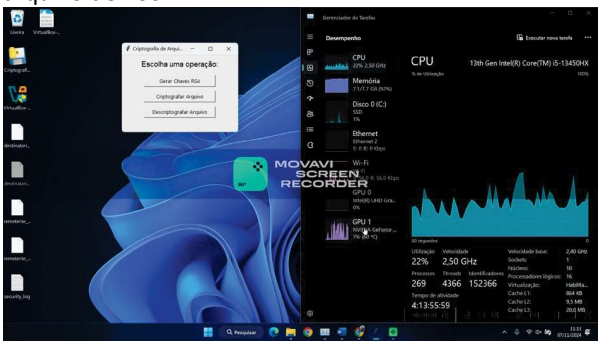
Fonte: os autores.

Testes semelhantes foram realizados em outro computador de características semelhantes, porém com um processador i5 de 13ª geração. Os resultados se deram conforme abaixo:

TABELA 6 – Segunda rodada de testes

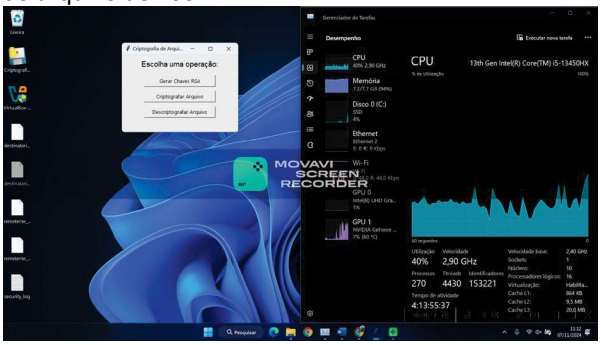
Tamanho do arquivo a ser cifrado	Tempo para cifrar	Tempo para decifrar
100MB	17 segundos	32 segundos
775MB	23 segundos	35 segundos

FIGURA 7 – Processamento durante a criptografia do arquivo de 100MB.



Fonte: os autores.

FIGURA 8 - Processamento durante a descriptografia do arquivo de 100MB.



Fonte: os autores.

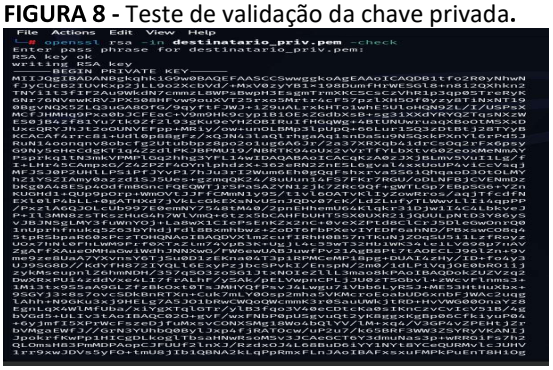
A ferramenta mostrou-se eficiente tanto na sua funcionalidade quanto no tempo de execução. As variações do funcionamento entre as máquinas foram percebidas dentro de uma margem aceitável.

3.3 TESTES DE SEGURANÇA

Além da avaliação do desempenho da aplicação, outro exame necessário é o da validade dos arquivos de chaves gerados pelo programa. Esta verificação de integridade permite a correta execução da ferramenta sem comprometimento da segurança.

Com o comando *openssl*, pode-se verificar se a chave está no formato correto e não corrompida, evitando falhas na operação.

FIGURA 8 - Teste de validação da chave privada.



Fonte: os autores.

Também é possível checar as informações e propriedades do arquivo de chave, garantindo que ele está pronto para uso.

FIGURA 9 - Teste de validação da chave pública

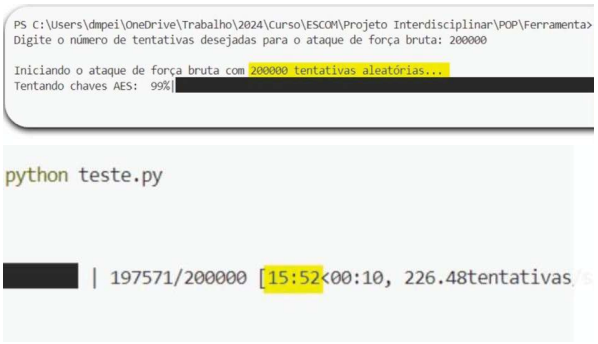
```
┌──(root㉿kali)-[/home/kali/chaves]
└─# openssl rsa -pubin -in destinatario_pub.pem -text -noout

Public-Key: (4096 bit)
Modulus:
    00:c1:d6:d7:e8:d9:1d:32:36:1c:0d:7c:9c:82:51:
    c0:76:21:4b:ca:c6:9d:a3:2c:bf:68:d9:77:1b:55:
    df:fe:33:15:74:cf:26:01:d7:1d:7d:f0:3b:a6:7c:
    7a:d6:11:21:a5:f3:e9:fc:8b:64:17:86:49:f6:4c:
    d6:22:d6:dd:df:20:5d:80:bb:d5:a4:74:de:dc:9a:
    6c:cb:f1:63:ec:07:0a:47:dc:4b:20:99:3a:e6:5c:
    a0:92:0a:c0:b3:56:14:75:a7:7a:a9:d1:24:eb:79:
    1c:8a:e8:da:fb:e8:d5:5e:c0:a4:55:24:f5:f9:d3:
    c1:c5:bf:0f:68:b9:75:53:db:9a:f1:a3:93:2b:b6:
    be:1c:17:9e:e9:ce:55:c7:e4:e7:f4:cb:3c:bc:4e:
    23:71:35:3d:7d:d0:18:2f:35:05:f9:64:b4:37:b8:
    60:3c:39:f1:bf:f6:ac:9f:b4:52:56:27:e8:99:f6:
    e0:0b:af:19:07:4e:8d:70:84:4e:54:96:81:d0:37:
    d6:4b:fc:8f:d4:48:fb:17:30:27:c9:1c:c1:ea:f4:
    fc:5a:d1:b2:42:14:46:82:f9:5f:66:f4:79:3d:73:
    2a:75:06:23:84:c5:91:9d:6d:7b:01:fa:c8:37:89:
    75:dd:61:16:10:65:3a:ac:35:7c:d6:11:2d:23:07:
    8c:df:f3:56:2e:ee:d9:3d:65:fd:a5:f7:78:0a:bb:
    d7:98:1d:93:81:21:1b:88:7c:71:aa:5a:0f:b8:06:
    d5:0d:53:0a:ee:6a:a5:c1:a0:eb:4c:49:7c:43:53:
    17:10:45:82:61:26:dd:a8:39:43:55:10:5a:69:f8:
    c4:62:cb:fa:30:fa:e9:ce:2c:13:29:de:5a:54:a5:
    0f:ba:e8:bb:ab:d5:24:37:cc:3b:41:b6:3d:bc:4d:
    8c:81:28:20:02:01:fe:2b:ad:cf:22:f9:47:65:d2:
    9f:3c:80:5c:ff:c5:02:4d:e3:79:5a:42:5a:e1:81:
    a0:2a:d6:c9:c3:69:2b:bd:35:24:31:90:f5:e7:62:
    5e:ab:3d:de:49:46:e3:75:e2:8a:27:aa:7b:fc:a1:
    b7:1f:83:65:2d:b9:b6:e9:cf:ca:68:da:8d:6e:83:
    a0:3a:26:bf:f6:6b:7e:d7:45:7a:9b:e2:27:6b:0a:
    c3:aa:da:b1:71:ea:9b:32:1b:d3:72:e5:e1:de:09:
    d8:0a:4f:5a:b8:67:37:65:3c:a2:41:3c:c5:35:f7:
    f3:41:45:32:bd:e2:85:31:da:f5:6b:4d:f6:0b:6d:
    7b:6f:eb:46:5e:a3:13:1e:36:60:18:3e:ca:6b:92:
    ad:6d:37:79:a4:54:f3:0f:94:6a:b6:86:18:37:60:
    52:f5:e3
Exponent: 65537 (0×10001)
```

Fonte: os autores.

A robustez da criptografia implementada na aplicação praticamente inviabiliza que os computadores mais avançados atualmente consigam tentar todas as combinações possíveis contra os algoritmos.

Executando um programa para realizar ataques de força bruta, foram determinadas 200.000 tentativas, as quais levaram cerca de 16 minutos para serem concluídas, sem sucesso.

FIGURA 9 – Ataque de força bruta
Fonte: os autores



Fonte: os autores.

Calcula-se, portanto, que para testar todas as combinações possíveis para quebrar um algoritmo AES-256, seriam necessários 1,65 x 10^59 anos.

Escalando a capacidade para um computador que possa realizar, por exemplo, 20 milhões de tentativas por segundo, o tempo médio cai para 1,835 x 10^51 anos, um número ainda inviável.

3.4 IMPLICAÇÕES PRÁTICAS E TEÓRICAS

Buscando conjugar elementos dos dois principais tipos de criptografia, a ferramenta desenvolvida trabalha com um modelo híbrido.

Dessa forma, é possível a transmissão de grandes quantidades de dados com segurança e eficiência, já que a funcionalidade simétrica garante boa velocidade na transmissão e a assimétrica protege a chave sem expô-la.

A obrigação de o usuário fornecer uma senha implementa também a autenticação de dois fatores, sendo um deles algo que o usuário tem (chave) e o outro algo que ele sabe (senha).

3.5 LIMITAÇÕES E CONSIDERAÇÕES

Embora a aplicação procure o melhor resultado ao combinar os algoritmos de ambos os tipos de criptografia, ainda é possível que a comunicação fique exposta a alguns riscos.

Ainda que as chaves estejam protegidas, a possível ausência de um canal de transmissão seguro entre os usuários, pode acarretar na interceptação das mesmas.

Se o usuário não inserir uma senha forte durante sua execução, ataques de força bruta podem ser suficientes para decodificá-la. Bem como, ela pode ser obtida por meio de técnicas de engenharia social.

Quanto ao seu funcionamento, apesar da sua boa execução, o programa não demonstra muita flexibilidade ao salvar todos os arquivos gerados no mesmo diretório local do executável. Futuras versões podem mitigar essa restrição.

3.6 RECOMENDAÇÕES FUTURAS

Os conhecimentos sobre a Segurança da Informação e a Proteção Cibernética estão sujeitos à constante revisão. Uma vez que a tecnologia segue evoluindo e novas ameaças são desenvolvidas, é crucial que o programa seja continuamente testado e sofra as devidas atualizações que se fizerem necessárias para garantir a inviolabilidade dos dados por ele protegidos.

Da mesma forma, é interessante que além de atualizada, a ferramenta possa ser integrada a outros sistemas utilizados pela Força Terrestre, inclusive com acesso à EBNet a fim de estabelecer um canal de comunicação seguro entre as partes.

ABSTRACT

This work aims to provide informations that can contribute to the development of data encryption and decryption application for the needs of the Brazilian Army's operations. Based on a summary analysis of the existing literature on the subject, the requirements for the development of a program that will implement encryption and decryption algorithms in a graphical interface will be defined, ensuring

security, performance and simplicity so that the user can protect his information from the various current cyber threats, in a efficient and safe way.

5 REFERÊNCIAS

BONEH, Dan. *Twenty Years of Attacks on the RSA Cryptosystem.* Notices of the American Mathematical Society, v. 46, n. 2, 1999.

DAEMEN, Joan; RIJMEN, Vincent. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Berlim: Springer-Verlag, 2001.

DIFFIE, Whitfield; LANDAU, Susan. *Privacy on the line: the politics of wiretapping and encryption*. Massachusetts: The MIT Press, 2007.

EXÉRCITO BRASILEIRO. *Doutrina Militar de Defesa Cibernética* (MD31-M-07). Brasília, 2023.

EXÉRCITO BRASILEIRO. *Diretriz Estratégica Organizadora do Sistema de Comando e Controle do Exército* (EB10-D-01.013). Brasília, 2021.

EXÉRCITO BRASILEIRO. *Manual de Guerra Cibernética* (EB70-MC-10.232). Brasília, 2017.

FERGUSON, Neil; SCHNEIER, Bruce; KOHNO, Tadayoshi. *Cryptography Engineering: Design Principles and Practice Applications*. New York: Wiley, 2010.

INTERNET ENGINEERING TASK FORCE (IETF). *RFC 1321: The MD5 message-digest algorithm*. Disponível em <https://www.ietf.org/rfc/rfc1321.txt> Acesso em: 28 out. 2024.

HELLMAN, Martin E. *An overview of public key cryptograph*. IEEE Communications Magazine, v. 16, n. 6, 1978.

KATZ, Jonathan; LINDELL, Yehuda. *Introduction to Modern Cryptography*. Boca Raton: CRC Press, 2007.

MENEZES, Alfred J.; VAN OORSCHOT, Paul C.; VANSTONE, Scott A. *Handbook of Applied Cryptography.* Boca Raton: CRC Press, 1996.

NAKAMURA, Emílio Tissato. GEUS, Paulo Lício de. Segurança de Redes em Ambientes Corporativos. São Paulo: Novatec Editora, 2007.

PAAR, Christof; PELZL, Jan. *Understanding Cryptography: A Textbook for Students and Practitioners.* Berlim: Springer-Verlag, 2010

SCHNEIER, Bruce. *Applied Cryptography*. New York: Wiley, 1996.

STALLINGS, William. *Criptografia e segurança de redes: Princípios e práticas*. São Paulo: Editora Pearson, 2015.

TANENBAUM, Andrew S. *Redes de computadores*. São Paulo: Editora Pearson Prentice Hall, 2011.

Brasília - DF, 8 de novembro de 2024.

RAPHAEL MACHADO DA SILVA RODRIGUES – Maj
Chefe da Seção de Ensino de TIC e Prot Ciber

APÊNDICE A – CONFECÇÃO E OPERAÇÃO DE UM PROGRAMA DE CRIPTOGRAFIA DE DADOS: PROCEDIMENTO OPERACIONAL PADRÃO (POP)

DESENVOLVIMENTO DE UMA FERRAMENTA DE CRIPTOGRAFIA DE DADOS: PRODUÇÃO DE UMA FERRAMENTA DE CRIPTOGRAFIA E DESCRIPTOGRAFIA PARA PROTEÇÃO DE DADOS ATRAVÉS DE UMA INTERFACE GRÁFICA

Cap **Flávio** Barros Correia
Cap Diego Madureira **Peixoto**
Cap Cassius Matheus Alves **Bierhals**

A.1 OBJETIVO

A.1.1 Fornecer diretrizes para a confecção, configuração e operação de um programa de criptografia de dados, incluindo instruções para o desenvolvimento e uso de um script de criptografia de dados, bem como padronizar o processo de utilização dele, garantindo a integridade, confidencialidade e disponibilidade das informações trocadas entre sistemas e unidades militares. Esse procedimento visa proporcionar uma camada de segurança adicional para prevenir acessos não autorizados e assegurar que as comunicações sejam protegidas contra ameaças cibernéticas durante operações.

A.2 ÁREA OU SETOR RESPONSÁVEL

A.2.1 Departamento de Tecnologia da Informação (TI).

A.3 REFERÊNCIAS

A.3.1 Documentação oficial da biblioteca PyCryptodome (https://pycryptodome.readthedocs.io/en/latest/src/api.html); e
A.3.2 Manual de Campanha EB70-MC-10.232 Guerra Cibernética, 1ª Edição, 2017.

A.4 MATERIAIS E EQUIPAMENTOS NECESSÁRIOS

A.4.1 Hardware compatível para servidor e estações de trabalho, com capacidade suficiente para processar operações de criptografia.
A.4.2 Python e as bibliotecas PyCryptodome para criptografia e Tkinter para interface gráfica.
A.4.3 Acesso à internet para atualização de bibliotecas e consulta de documentação online, se necessário.
A.4.4 Documentação técnica sobre criptografia, contendo especificações de segurança e manuais de referência para configuração e operação segura.
A.4.6 Ferramentas de backup e recuperação de dados para garantir a integridade e a disponibilidade das informações em caso de falha no sistema.

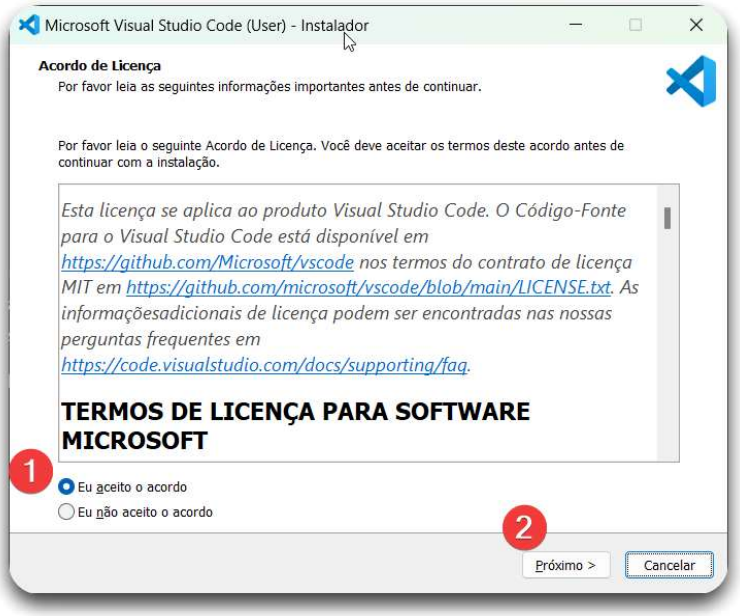
A.5 PROCEDIMENTOS PASSO A PASSO

A.5.1 INSTALAÇÃO

A.5.1.1 Preparação do Ambiente

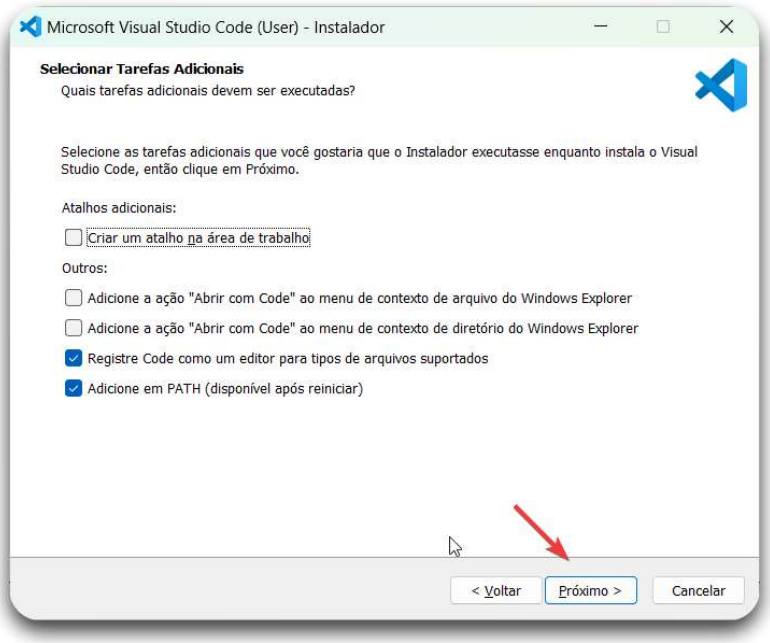
A.5.1.1.1: Para melhor visualização do código e por ocasião deste POP, será utilizado o editor de código fonte Visual Studio Code (https://code.visualstudio.com/download)

FIGURA 1 – Passo 1 da instalação do programa



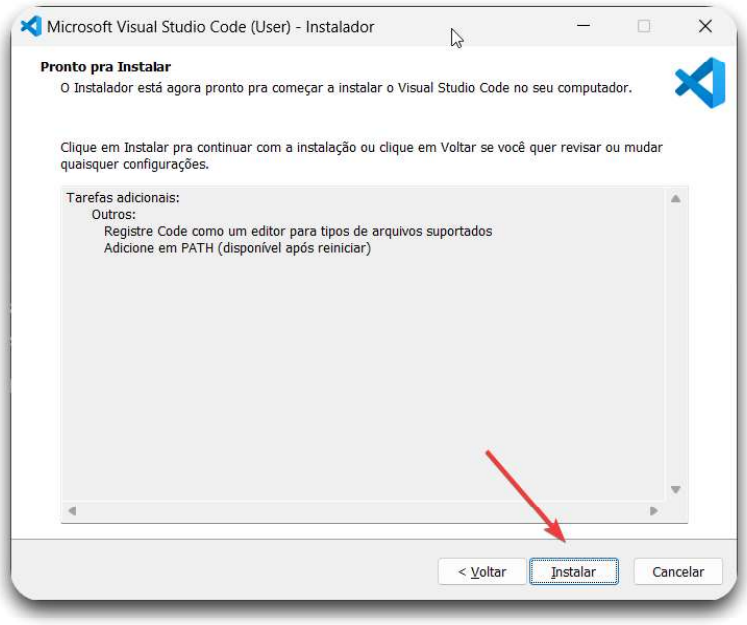
Fonte: os autores.

FIGURA 2 - Passo 2 da instalação do programa



Fonte: os autores.

FIGURA 3 - Passo 3 da instalação do programa



Fonte: os autores.

DESCRIÇÃO DO PASSO: *Para instalar o Visual Studio Code, aceite o acordo de licença e clique em "Próximo". Marque as opções desejadas, incluindo criar um atalho e adicionar ao PATH, e clique em "Próximo". Confirme as configurações e clique em "Instalar" para iniciar.*
No Linux também pode ser baixado via terminal através do comando: sudo apt update && sudo apt install code.

A.5.1.1.2 Instalar o Python, que será a linguagem de programação utilizada para a confecção do script de criptografia (https://www.python.org/downloads/)

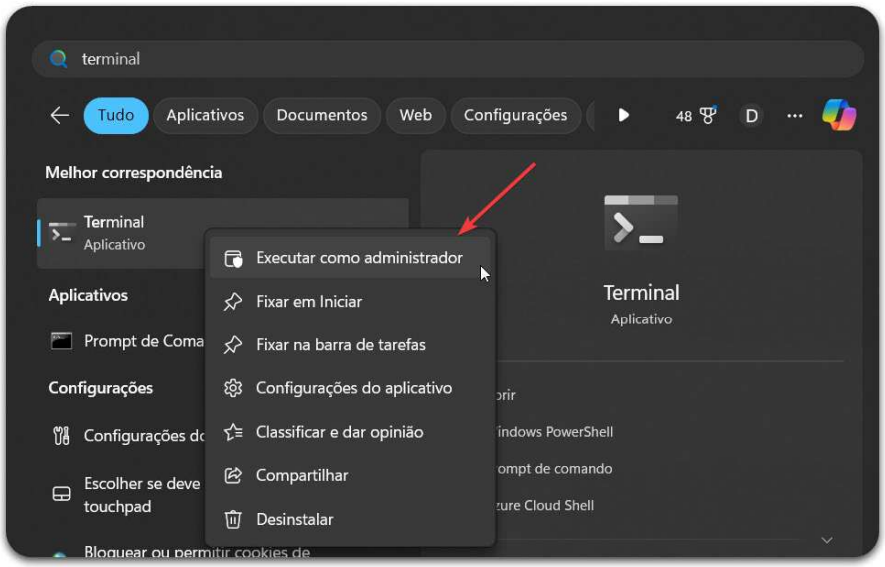
FIGURA 4 – Instalação do Python

Fonte: os autores

DESCRIÇÃO DO PASSO: *Para instalar o Python, marque a opção "Use admin privileges when installing py.exe" para garantir permissões administrativas e "Add python.exe to PATH" para adicionar o Python ao PATH do sistema. Em seguida, clique em "Install Now" para iniciar a instalação com as configurações padrão.*
No Linux, a instalação do Python pode ser feita diretamente pelo terminal. Use os seguintes comandos: sudo apt update && sudo apt install python3

A.5.1.1.3 Instalação das bibliotecas de python necessárias para confecção do script:

FIGURA 5 – Instalação das bibliotecas



Fonte: os autores

DESCRIÇÃO DO PASSO: Abra o terminal e execute: no Windows, ***pip install pycryptodome pyinstaller*** (o Tkinter já vem com o Python); no Linux, ***pip install pycryptodome pyinstaller*** e ***sudo apt-get install python3-tk***

A.5.2 CONFECÇÃO DO SCRIPT

A.5.2.1 Script de Criptografia Híbrida

A.5.2.1.1 Dentro do Visual Studio Code, crie um arquivo "***.py***" onde será inserido o script para criptografia híbrida. O script completo será disponibilizado no final do pop:

FIGURA 6 – Trecho do script

```
from tkinter import Tk, Label, Button, filedialog, messagebox, simpledialog
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP, AES
from Crypto.Random import get_random_bytes
from Crypto.Hash import SHA256
from Crypto.Signature import pkcs1_15
import base64
import os
from datetime import datetime
```

Fonte: os autores

DESCRIÇÃO DO PASSO: Primeiramente serão realizadas as importações de bibliotecas essenciais para o funcionamento do script de criptografia. O ***tkinter*** fornece componentes de interface gráfica, como botões e caixas de diálogo. A biblioteca ***Crypto*** (do pacote PyCryptodome) importa módulos para geração e uso de chaves RSA, criptografia AES, geração de bytes aleatórios, e criação de hashes e assinaturas digitais. ***base64*** é usada para codificação de dados, ***os*** para operações no sistema, e ***datetime*** para registros de data e hora em logs ou relatórios.

A.5.2.1.2 Funções de utilidade dentro no script de criptografia:

FIGURA 7 – Funções de utilidade

```
def log_action(action, status, file_path=""):
    with open("security_log.txt", "a") as log_file:
        log_file.write(f"[{datetime.now()}] - {action}: {status} - {file_path}\n")

def show_message(success, msg, action="", file_path=""):
    if success:
        messagebox.showinfo("Sucesso", msg)
    else:
        messagebox.showerror("Erro", msg)
    log_action(action, "Sucesso" if success else "Falha", file_path)

def prompt_password(prompt_text="Digite a senha:"):
    return simpledialog.askstring("Senha", prompt_text, show='*')

def load_key(file_path, key_type):
    try:
        with open(file_path, "rb") as key_file:
            key_data = key_file.read()
            if key_type == "private":
                password = prompt_password("Digite a senha da chave privada:")
                if not password:
                    raise ValueError("Senha não fornecida.")
                key = RSA.import_key(key_data, passphrase=password)
            else:
                key = RSA.import_key(key_data)
            if key_type == "public" and key.has_private():
                raise ValueError("Esperava chave pública, mas uma chave privada foi fornecida.")
            if key_type == "private" and not key.has_private():
                raise ValueError("Esperava chave privada, mas uma chave pública foi fornecida.")
            return key
    except Exception as e:
        show_message(False, f"Erro ao carregar chave {key_type}: {e}", f"Carregar chave {key_type}", file_path)
        return None
```

Fonte: os autores

DESCRIÇÃO DO PASSO: Essas funções de utilidade ajudam no gerenciamento do script. A função ***log_action*** registra as ações em um arquivo de log com data e hora; ***show_message*** exibe mensagens de sucesso ou erro para o usuário e registra a ação; ***prompt_password*** solicita uma senha ao usuário; e ***load_key*** carrega uma chave RSA (privada ou pública) do arquivo, pedindo senha se necessário e verificando se o tipo de chave está correto, garantindo segurança e usabilidade.

A.5.2.1.3 Função para gerar as chaves RSA:

FIGURA 8 – Função generate_rsa_keys()

```
def generate_rsa_keys():
    try:
        # Pedir o nome da chave
        key_name = simpledialog.askstring("Nome da Chave", "Escolha um nome para as chaves RSA:")
        if not key_name:
            show_message(False, "Nome da chave não pode estar vazio.", "Gerar chave RSA")
            return

        password = None
        while True:
            password = prompt_password("Digite uma senha para proteger a chave privada:")
            if password:
                confirm_password = prompt_password("Confirme a senha:")
                if password == confirm_password:
                    break
                else:
                    show_message(False, "As senhas não coincidem. Tente novamente.", "Gerar chave RSA")
            else:
                show_message(False, "Senha não pode estar vazia. Tente novamente.", "Gerar chave RSA")

        # Gerar as chaves RSA
        key = RSA.generate(4096)
        private_key_filename = f"{key_name}_priv.pem"
        public_key_filename = f"{key_name}_pub.pem"

        with open(private_key_filename, "wb") as priv_file, open(public_key_filename, "wb") as pub_file:
            priv_file.write(key.export_key(passphrase=password, pkcs=8, protection="scryptAndAES128-CBC"))
            pub_file.write(key.publickey().export_key())

        show_message(True, f"Chaves RSA geradas e salvas como {private_key_filename} e {public_key_filename}.", "Gerar chave RSA")
    except Exception as e:
        show_message(False, f"Erro ao gerar chaves RSA: {e}", "Gerar chave RSA")
```

Fonte: os autores

DESCRIÇÃO DO PASSO: A função ***generate_rsa_keys*** cria um par de chaves RSA (privada e pública). Primeiro, pede ao usuário um nome para as chaves e uma senha para proteger a chave privada. A senha é confirmada para garantir precisão. As chaves são geradas com 4096 bits e salvas em arquivos com nomes baseados na escolha do usuário. Em caso de sucesso ou erro, uma mensagem é exibida ao usuário, informando o resultado da operação.

A.5.2.1.4 Função para criptografia:

FIGURA 9 – Função encrypt_file()

```
def encrypt_file(public_key_path, private_key_path, file_path):
    try:
        public_key = load_key(public_key_path, "public")
        private_key = load_key(private_key_path, "private")
        if not public_key or not private_key:
            return

        aes_key = get_random_bytes(32)
        with open(file_path, "rb") as f:
            file_data = f.read()

        cipher_aes = AES.new(aes_key, AES.MODE_GCM)
        ciphertext, tag = cipher_aes.encrypt_and_digest(file_data)
        encrypted_aes_key = PKCS1_OAEP.new(public_key).encrypt(aes_key)
        signature = pkcs1_15.new(private_key).sign(SHA256.new(ciphertext))

        with open(f"{file_path}.enc", "wb") as enc_file:
            for item in [cipher_aes.nonce, encrypted_aes_key, ciphertext, tag, signature, os.path.splitext(file_path)[1].encode()]:
                enc_file.write(base64.b64encode(item) + b'\n')

        show_message(True, f"Arquivo '{file_path}' criptografado com sucesso.", "Criptografar arquivo", file_path)

    except Exception as e:
        show_message(False, f"Erro durante a criptografia: {e}", "Criptografar arquivo", file_path)
```

Fonte: os autores

DESCRIÇÃO DO PASSO: A função ***encrypt_file*** criptografa um arquivo usando uma chave AES gerada aleatoriamente e criptografa essa chave AES com a chave pública RSA. Ela também gera uma assinatura digital com a chave privada RSA para garantir a integridade. O arquivo criptografado é salvo com extensão ***.enc*** e inclui a chave AES criptografada, o texto cifrado, a tag de autenticação, a assinatura e a extensão original do arquivo. Em caso de sucesso ou erro, uma mensagem é exibida ao usuário com o status da operação.

A.5.2.1.5 Função para descriptografia:

FIGURA 10 – Função decrypt_file()

```
def decrypt_file(public_key_path, private_key_path, file_path):
    try:
        public_key = load_key(public_key_path, "public")
        private_key = load_key(private_key_path, "private")
        if not public_key or not private_key:
            return

        with open(file_path, "rb") as enc_file:
            nonce, encrypted_aes_key, ciphertext, tag, signature, file_extension = [base64.b64decode(enc_file.readline().strip()) for _ in range(6)]

        aes_key = PKCS1_OAEP.new(private_key).decrypt(encrypted_aes_key)
        cipher_aes = AES.new(aes_key, AES.MODE_GCM, nonce=nonce)
        decrypted_data = cipher_aes.decrypt_and_verify(ciphertext, tag)
        pkcs1_15.new(public_key).verify(SHA256.new(ciphertext), signature)

        output_file_path = filedialog.asksaveasfilename(title="Salvar arquivo descriptografado como", defaultextension=file_extension.decode())
        if output_file_path:
            with open(output_file_path, "wb") as out_file:
                out_file.write(decrypted_data)
            show_message(True, f"Arquivo '{output_file_path}' descriptografado com sucesso.", "Descriptografar arquivo", output_file_path)

    except Exception as e:
        show_message(False, f"Erro durante a descriptografia: {e}", "Descriptografar arquivo", file_path)
```

Fonte: os autores

DESCRIÇÃO DO PASSO: A função ***decrypt_file*** descriptografa um arquivo usando a chave privada RSA para recuperar a chave AES e, em seguida, descriptografa o conteúdo do arquivo com AES. Ela também verifica a assinatura digital com a chave pública para assegurar a integridade do arquivo. Após a descriptografia, o usuário escolhe onde salvar o arquivo restaurado com sua extensão original. Em caso de sucesso ou erro, uma mensagem informa o status da operação ao usuário.

A.5.2.1.6 Funções para interface gráfica:

FIGURA 11 – Funções para interface gráfica

```
def encrypt_file_ui():
    public_key = filedialog.askopenfilename(title="Selecione a chave pública do destinatário", filetypes=[("Chave Pública", "*.pem")])
    private_key = filedialog.askopenfilename(title="Selecione a sua chave privada", filetypes=[("Chave Privada", "*.pem")])
    file_path = filedialog.askopenfilename(title="Selecione o arquivo para criptografar", filetypes=[("Todos os arquivos", "*.*")])
    if public_key and private_key and file_path:
        encrypt_file(public_key, private_key, file_path)

def decrypt_file_ui():
    private_key = filedialog.askopenfilename(title="Selecione a sua chave privada", filetypes=[("Chave Privada", "*.pem")])
    public_key = filedialog.askopenfilename(title="Selecione a chave pública do remetente", filetypes=[("Chave Pública", "*.pem")])
    encrypted_file = filedialog.askopenfilename(title="Selecione o arquivo criptografado", filetypes=[("Arquivo Criptografado", "*.enc")])
    if private_key and public_key and encrypted_file:
        decrypt_file(public_key, private_key, encrypted_file)

def create_gui():
    root = Tk()
    root.title("Criptografia de Arquivos")
    Label(root, text="Escolha uma operação:", font=("Arial", 14)).pack(pady=10)
    Button(root, text="Gerar Chaves RSA", command=generate_rsa_keys, width=25).pack(pady=5)
    Button(root, text="Criptografar Arquivo", command=encrypt_file_ui, width=25).pack(pady=5)
    Button(root, text="Descriptografar Arquivo", command=decrypt_file_ui, width=25).pack(pady=5)
    root.geometry("300x200")
    root.mainloop()
```

Fonte: os autores

DESCRIÇÃO DO PASSO: Essas funções criam a interface gráfica para criptografia de arquivos. ***encrypt_file_ui*** e ***decrypt_file_ui*** permitem que o usuário selecione as chaves e o arquivo para criptografar ou descriptografar usando caixas de diálogo. A função ***create_gui*** configura a janela principal com botões para gerar chaves RSA, criptografar e descriptografar arquivos, tornando o processo acessível e fácil de usar.

A.5.2.1.7 Função para execução do programa:

FIGURA 12 – Função principal

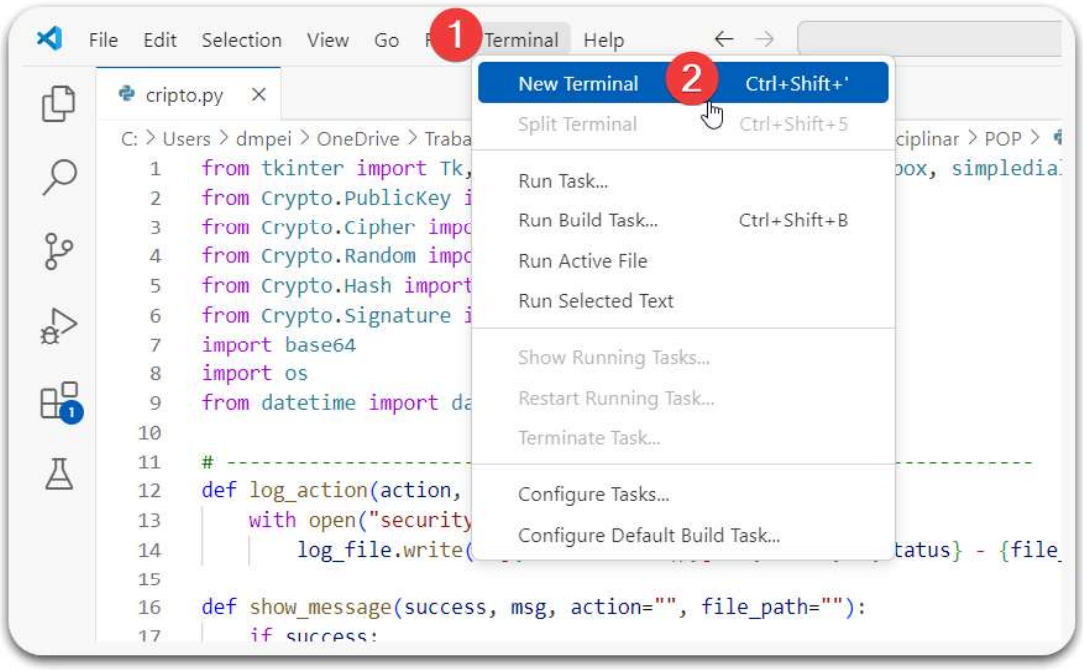
```
if __name__ == "__main__":
    create_gui()  # Inicia a interface gráfica
```

Fonte: os autores

DESCRIÇÃO DO PASSO: A condição ***if __name__ == "__main__"***: verifica se o script está sendo executado diretamente. Se for o caso, a função ***create_gui()*** é chamada, iniciando a interface gráfica do usuário para acessar as funcionalidades de criptografia e descriptografia de arquivos.

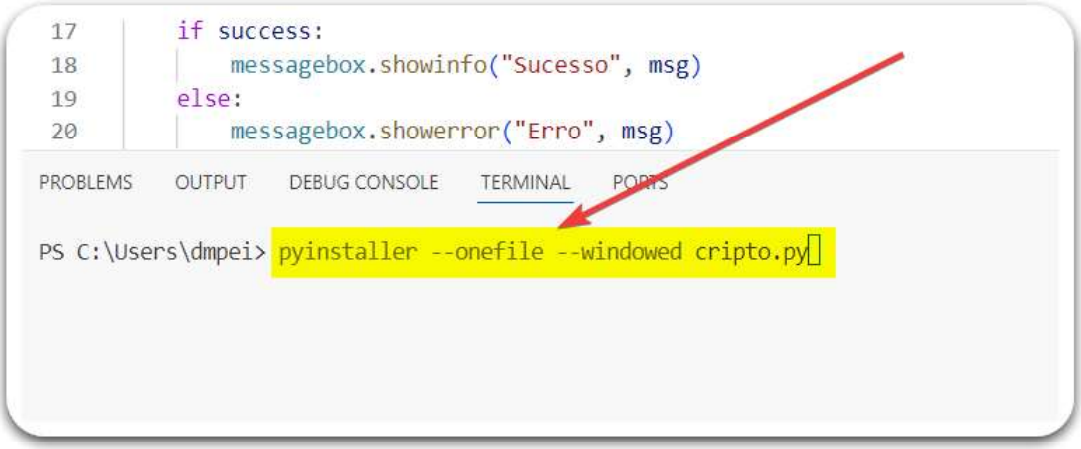
A.5.2.1.8 Importando o script em um executável:

FIGURA 13 – Passo 1 da importação do script



Fonte: os autores

FIGURA 12 - Passo 2 da importação do script



Fonte: os autores

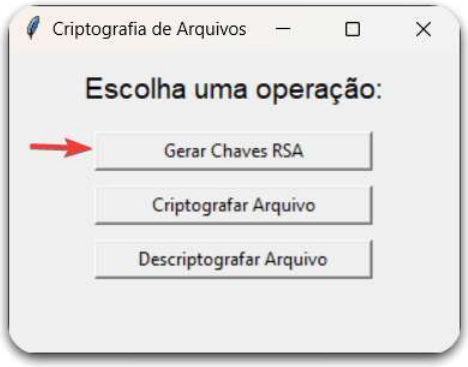
DESCRIÇÃO DO PASSO: Para exportar o script como executável no Visual Studio Code, abra o terminal integrado, navegue até o diretório do script e execute ***pyinstaller --onefile --windowed nome_do_script.py*** no terminal. O executável será gerado na pasta ***dist*** dentro do diretório do projeto, pronto para uso sem necessidade de um ambiente Python.
Nota: O executável gerado pelo PyInstaller funcionará apenas no sistema operacional onde o comando foi executado. Para criar versões para Windows e Linux, é necessário rodar o comando em cada sistema operacional.

A.5.3 OPERAÇÃO

A.5.3.1 Gerando Chaves RSA

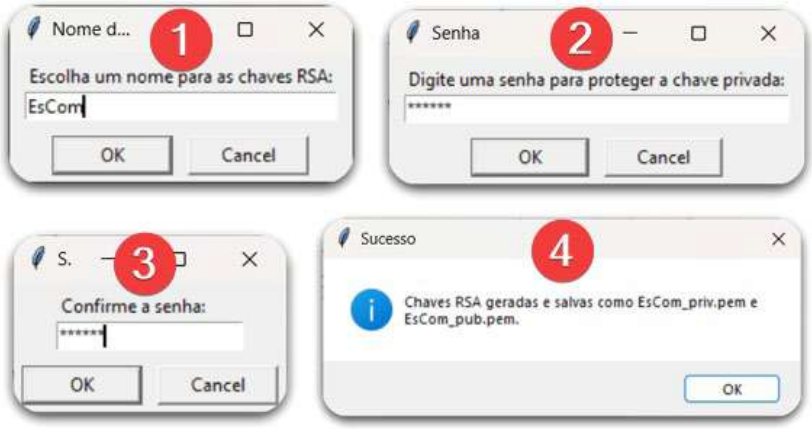
A.5.3.1.1 Gerar as chaves públicas e privadas que irão servir para garantir a integridade e autenticidade dos dados:

FIGURA 13 – Seleção da opção “gerar chaves RSA” no programa



Fonte: os autores.

FIGURA 14 - Passo a passo da geração de chaves



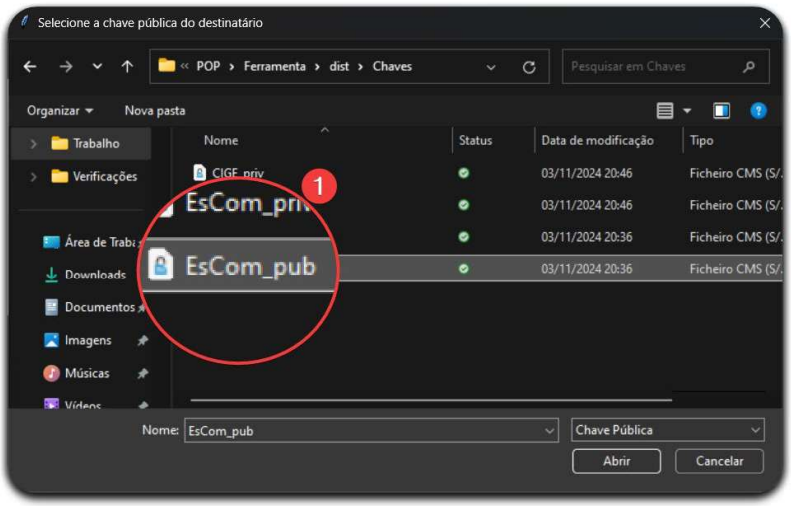
Fonte: os autores.

DESCRIÇÃO DO PASSO: Clique em "***Gerar Chaves RSA***", insira um nome para o par de chaves e defina uma senha para proteger a chave privada, confirmando-a em seguida. As chaves serão salvas na mesma pasta do programa com os nomes escolhidos (***ex: EsCom_priv.pem e EsCom_pub.pem***).

A.5.3.2 Criptografando arquivos

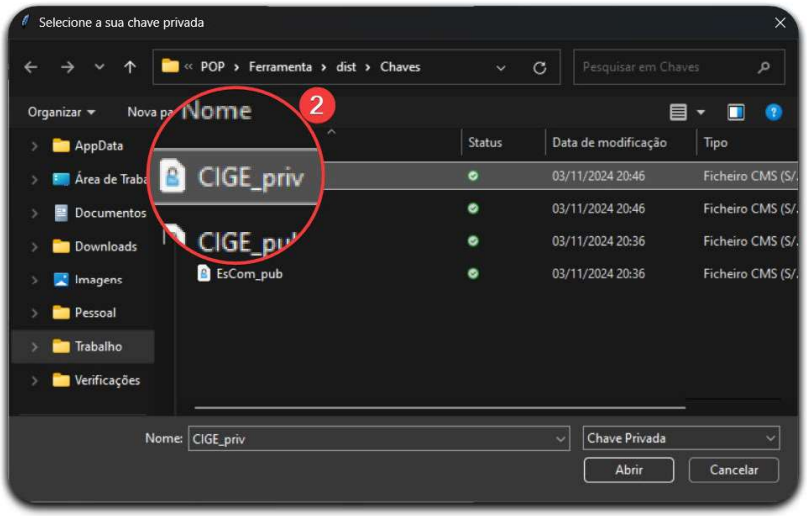
A.5.3.2.1 Este passo utiliza criptografia para proteger o arquivo selecionado com as chaves RSA;

FIGURA 15 – Seleção da chave pública



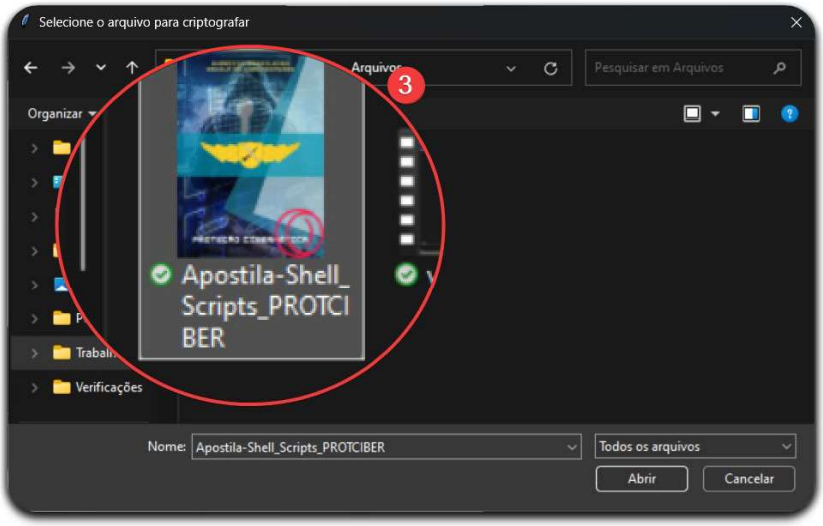
Fonte: os autores.

FIGURA 16 - Seleção da chave privada



Fonte: os autores.

FIGURA 17 - Seleção do arquivo a ser cifrado



Fonte: os autores.

FIGURA 18 – Digitação da senha e mensagem final

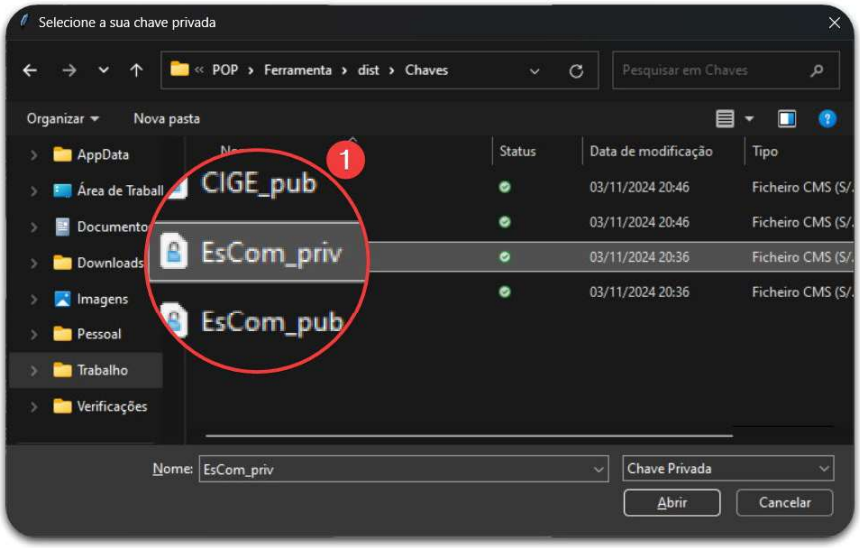
Fonte: os autores.

DESCRIÇÃO DO PASSO: Para criptografar um arquivo, clique em "***Criptografar Arquivo***" e selecione a chave pública do destinatário (***EsCom_pub.pem***). Em seguida, escolha sua chave privada (***CIGE_priv.pem***) e o arquivo que deseja criptografar. Insira a senha da chave privada quando solicitado. Ao final, uma mensagem de sucesso confirmará que o arquivo foi criptografado, e a versão criptografada será salva na mesma pasta do arquivo original, com a extensão "***. enc***".

A.5.3.3 Descriptografando arquivos

A.5.3.2.1 Este passo reverte a criptografia, restaurando o arquivo original.;

FIGURA 19 – Seleção da chave privada

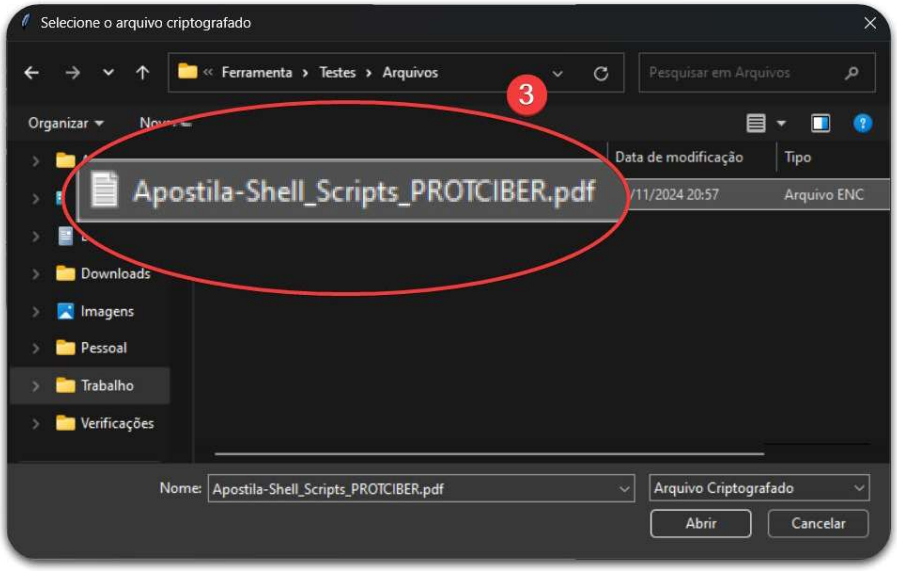


Fonte: os autores.

FIGURA 20 - Seleção da chave pública

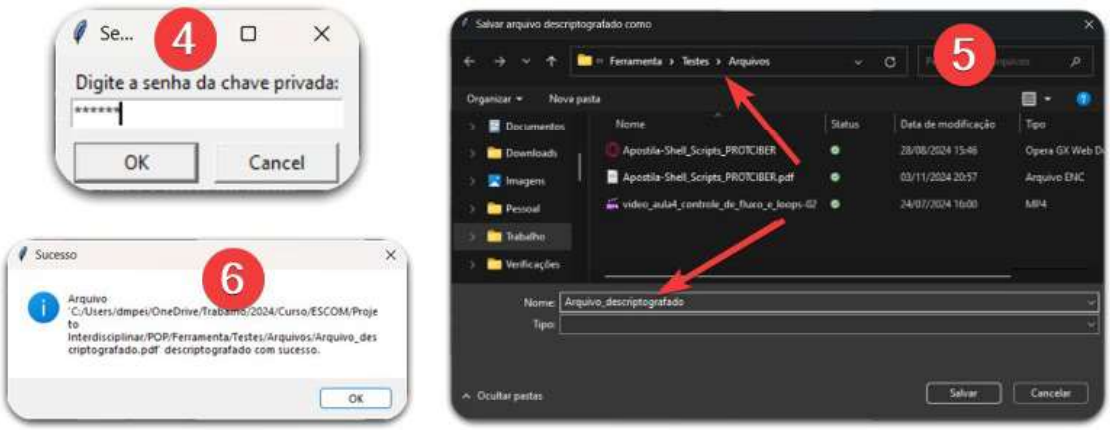
Fonte: os autores.

FIGURA 21 - Seleção do arquivo a ser decifrado



Fonte: os autores.

FIGURA 22 - Digitação da senha e mensagem final



Fonte: os autores.

DESCRIÇÃO DO PASSO: Para descriptografar um arquivo, clique em "***Descriptografar Arquivo***" e selecione sua chave privada (***EsCom_priv.pem***) e a chave pública do remetente (***CIGE_pub.pem***). Escolha o arquivo criptografado (tipo ***enc***) e, em seguida, insira a senha da chave privada quando solicitado. Defina o local e o nome para salvar o arquivo descriptografado e confirme. Uma mensagem de sucesso indicará que o arquivo foi restaurado com sucesso no local escolhido.

A.6 CONTROLES DE QUALIDADE

A.6.1 Realizar testes de criptografia e descriptografia após a geração das chaves e configuração para garantir que o programa está operando corretamente.

A.6.2 Verificar a integridade dos arquivos criptografados e descriptografados para assegurar que não houve perda de dados durante o processo.

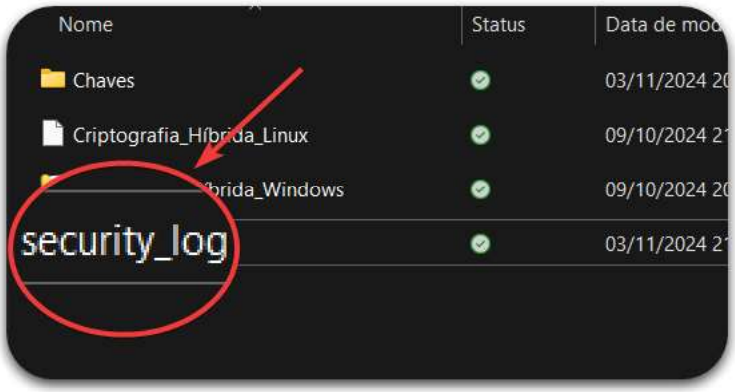
A.6.3 Realizar testes de compatibilidade das chaves RSA, assegurando que somente a chave privada correspondente pode descriptografar os arquivos.

A.6.4 Testar a interface gráfica para garantir que todas as funcionalidades (geração de chaves, criptografia, descriptografia) estão operando sem erros.

A.7 REGISTROS E DOCUMENTAÇÃO

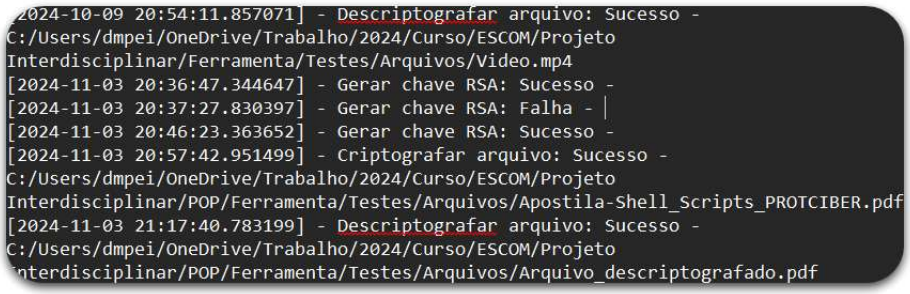
A.7.1 Manter registros de todas as etapas do processo de geração de chaves, criptografia e descriptografia de arquivos e documentar qualquer problema encontrado durante o uso do programa, como falhas na criptografia ou erros de autenticação.

FIGURA 23 – Arquivo de log gerado



Fonte: os autores.

FIGURA 24 – Conteúdo do arquivo de log



Fonte: os autores.

DESCRIÇÃO DO PASSO: O log do programa, criado automaticamente no arquivo security_log.txt na mesma pasta onde o programa é executado, registra todas as ações realizadas, como geração de chaves, operações de criptografia e descriptografia, além de eventuais falhas. Cada entrada inclui a data, a ação executada, o status (sucesso ou falha) e, quando aplicável, o caminho do arquivo envolvido.

A.8 RESPONSABILIDADES

A.8.1 O responsável pela TI é responsável por supervisionar a implementação do programa de acordo com este POP; e

A.8.2 Os usuários finais são responsáveis por reportar quaisquer problemas ou necessidades de suporte relacionadas ao uso do programa de criptografia híbrida.

A.9 ANEXOS

A.9.1 Script do programa completo, em formato .py