IVAN MEDINA

TIAGO GOTO SALA

mtCrypt

APLICATIVO PARA ARMAZENAMENTO SEGURO DE DADOS EM DISPOSITIVOS MÓVEIS

SÃO PAULO

2015

IVAN MEDINA

TIAGO GOTO SALA

mtCrypt

APLICATIVO PARA ARMAZENAMENTO SEGURO DE DADOS EM DISPOSITIVOS MÓVEIS

Área de Concentração:

Engenharia de Computação e Sistemas Digitais

Orientador:

Prof. Dr. Marcos Antonio Simplicio Junior

SÃO PAULO

2015

FICHA CATALOGRÁFICA

Medina, Ivan

mtCrypt: APLICATIVO PARA ARMAZENAMENTO SEGURO DE DADOS EM DISPOSITIVOS MÓVEIS / I. Medina, T. G. Sala -- São Paulo, 2015.

95 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Segurança 2.Criptografia 3.Android 4.TrueCrypt I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Sala, Tiago Goto

RESUMO

Este trabalho visa implementar mecanismos seguros de armazenamento de dados em dispositivos móveis. Além disso, tem-se por objetivo proteger o usuário e seus dados confidenciais em situações onde ele é forçado a revelar sua senha. Para isto, propõe-se a utilização dos conceitos de *deniable encrypion* em dispositivos móveis que operam sobre o sistema operacional Android. O aplicativo final a ser desenvolvido será baseado em uma plataforma já existente, o TrueCrypt, e apresentará total compatibilidade com a mesma.

Palavras-chave: dispositivos móveis, segurança, criptografia, criptografia negável, Android, TrueCrypt.

ABSTRACT

This work aims to implement safe data storage mechanisms on mobile devices. In addition, it has been designed to protect its user and his sensitive data in situations where he is forced to reveal his password. For that, we propose to use the concepts of deniable encrypion on mobile devices that run the Android operating system. The final application to be developed will be based on an existing platform, TrueCrypt, and will provide full compatibility with it.

Palavras-chave: mobile devices, security, cryptography, deniable encryption, Android, TrueCrypt.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AES	*Advanced Encryption Standard*
ASCII	*American Standard Code for Information Interchange*
CPS	*Ciphertext-Stealing*
CRC	*Cyclic Redundancy Check*
ECB	*Electronic Code Book*
FAT	*File Allocation Table*
FUSE	*Filesystem in Userspace*
GF	*Galois Field*
IDE	*Integrated Development Environment*
ISO	*International Organization for Standardization*
MBR	Master Boot Record
NTFS	*New Technology File System*
RIPEMD	*Race Integrity Primitives Evaluation Message Digest*
SDK	*Software Development Kit*
SHA	*Secure Hash Algorithm*
XEX	*Xor-Encrypt-Xor*
XTS	*XEX encryption mode with tweak and ciphertext stealing*

SUMÁRIO

1 INTRODUÇÃO

Os dispositivos móveis estão sendo amplamente adotados pelos usuários, ocupando um espaço cada vez maior no cotidiano dos mesmos. Com isto, uma quantidade cada vez maior de dados, tanto pessoais quanto de natureza corporativa, são armazenados nestes dispositivos.

Apesar de a maioria dos fabricantes proporcionar algum tipo de mecanismo de segurança para proteger os dados contidos no dispositivo, eles se mostram ineficazes em situações onde o usuário é coagido a revelar suas senhas. Neste cenário, mecanismos de *deniable encryption* fornecem proteção ao usuário, possibilitando-o fornecer uma senha falsa que revele apenas parte dos dados, protegendo os dados mais sensíveis.

Vislumbrando a necessidade desse tipo de funcionalidade nos dispositivos móveis, este trabalho propõe implementar um meio de solucionar o problema descrito acima.

1.1 Objetivo

O objetivo deste trabalho é desenvolver uma aplicação voltada a dispositivos móveis da plataforma Android, que forneça ao usuário mecanismos de *deniable encryption*. Para isto, a aplicação irá se basear em uma outra solução já existente para computadores pessoais, o TrueCrypt.

O aplicativo final, nomeado de mtCrypt, será desenvolvido para dispositivos que operem sobre a plataforma Android e proporcionará total compatibilidade com o TrueCrypt. Dessa forma, um usuário poderá intercambiar livremente seus volumes criptografados entre seu computador pessoal e seu dispositivo móvel.

1.2 Justificativa

Os dispositivos móveis proporcionam uma plataforma que possibilita o fácil acesso e transporte do mais diversos tipos de dados. Seu uso crescente nas mais diversas situações faz com que a proteção dos dados nestes contidos seja de importância cada vez maior.

A proteção fornecida pelos mecanismos padrões incluídos pelos fabricantes dos dispositivos móveis se mostram insuficientes em situações onde o proprietário do dispositivo é obrigado a fornecer suas senhas. Alguns exemplos que ilustram esse tipo de situação são:

- Apreensão do dispositivo por agentes governamentais em nações onde existem leis específicas que obriguem o usuário a revelar seus dados;
- Zonas de conflito, onde jornalistas e ativistas de direitos humanos utilizam dispositivos móveis para transportar dados que comprovem ações ilegais dos agentes locais;
- Coerção com utilização de violência ou ameaças.

Tais situações exigem mecanismos que protejam os dados mesmo após a revelação forçada de uma senha. É neste tipo de cenário que uma aplicação como o mtCrypt se mostra necessária e de grande utilidade, pois possibilita um acesso parcial dos dados, mantendo em sigilo as informações mais críticas.

1.3 Organização do documento

Este documento está dividido da seguinte forma:

- Seção 1 - Introdução: apresenta um panorama geral do uso de dispositivos móveis no contexto da proteção dos dados contidos nos mesmos. Além disso, são

apresentados os objetivos e a justificativa ao desenvolvimento da solução frente ao cenário de utilidade apresentado;

- Seção 2 - Aspectos Conceituais: são descritos alguns conceitos importantes na área de criptografia, úteis para a compreensão das tecnologias utilizadas e funções implementadas pelo aplicativo. Nesta seção também é explorada a aplicação TrueCrypt, detalhando sua forma de operação a fim de estabelecer a compatibilidade com o mtCrypt;
- Seção 3 - Especificação: são descritos os requisitos que o sistema deve implementar, os casos de uso identificados, alguns modelos do sistema e a interface do aplicativo;
- Seção 4 - Implementação: são descritos as tecnologias, os métodos e arquivos auxiliares utilizados no desenvolvimento da solução, bem como os códigos desenvolvidos e os artefatos finais produzidos.

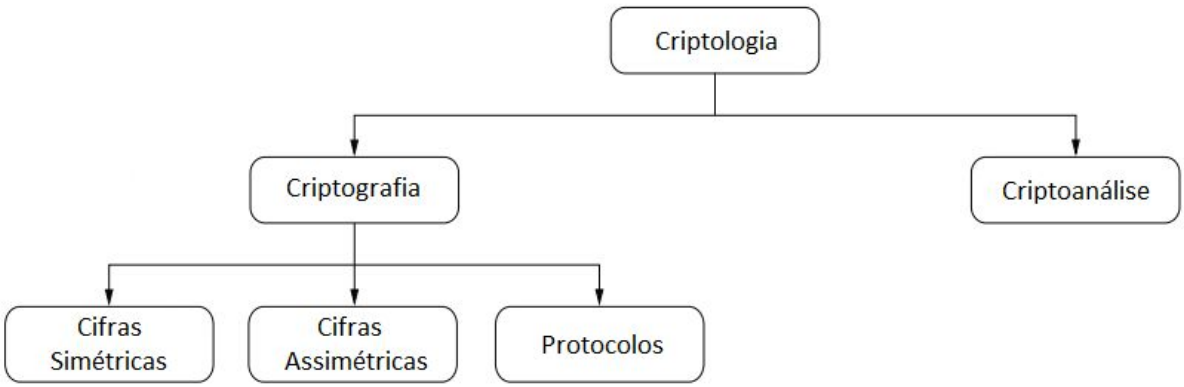
2 ASPECTOS CONCEITUAIS

Nesta seção, são apresentados alguns aspectos conceituais relevantes ao tema deste trabalho. O objetivo é estabelecer um nível de conceituação suficiente para o entendimento das particularidades técnicas e teóricas envolvidas.

2.1 Criptografia

A criptografia é a ciência que estuda princípios e técnicas de transformação de informações com o objetivo de esconder seu significado de destinatários indesejados. Está inserida dentro da área de Criptologia, junto com a Criptoanálise. A Figura 2.1 abaixo demonstra essa estrutura:

Figura 2.1 - Criptologia



Fonte: Adaptado de [5], p. 3.

Dentro da Criptografia, existem três grandes ramificações:

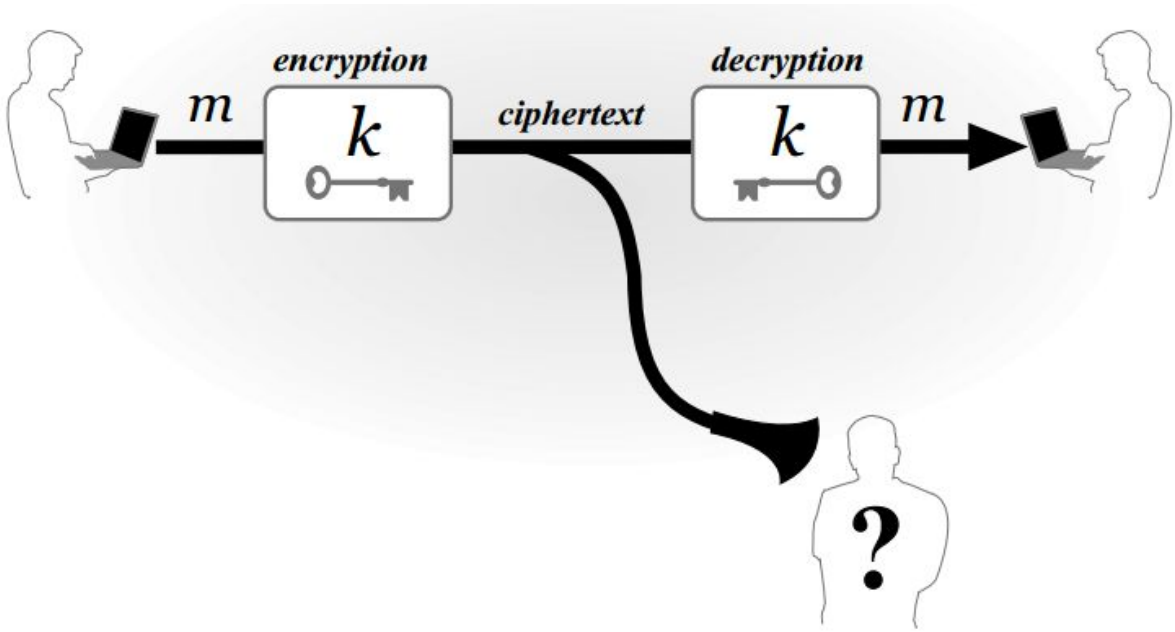
- Cifras simétricas: algoritmos de encriptação e decriptação em que as partes envolvidas compartilhas uma chave secreta;
- Cifras assimétricas: algoritmos de encriptação e decriptação em que existem duas chaves, uma pública e uma privada;
- Protocolos: aplicação dos algoritmos criptográficos nas comunicações de rede.

Este trabalho só lida com a criptografia no ramo das cifras simétricas e, portanto, segue a seguir na seção 2.1.1 uma explicação mais detalhada sobre o tema.

2.1.1 Criptografia simétrica (chave privada)

Como descrito anteriormente, as cifras simétricas são algoritmos de encriptação e decriptação em que as partes envolvidas compartilhas uma chave secreta. Essa chave é utilizada para transformar uma informação base em texto cifrado (processo denominado encriptação), e também serve para converter o texto cifrado de volta para a informação base (processo denominado decriptação). A Figura 2.2 abaixo representa esse processo:

Figura 2.2 - Criptografia simétrica



Fonte: [7], p. 5.

Nesta imagem, um terceiro indivíduo que não possui a chave não consegue traduzir o texto cifrado e, portanto, não consegue extrair a informação da mensagem que está sendo transmitida.

E equação genérica que representa essa transformação é:

$$Dec_k(Enc_k(m)) = m$$

Onde:

- Dec função de decriptação da mensagem;
- Enc função de encriptação da mensagem;
- k chave;
- m mensagem.

As cifras simétricas podem ser divididas em dois grandes grupos:

- Cifras de fluxo: consistem em algoritmos de geração de uma sequência de bits pseudoaleatória (dependente da chave fornecida), que se combina com os dados originais, normalmente por uma operação "*xor*";
- Cifras de bloco: consistem em algoritmos que operam sobre um bloco de dados de tamanho pré-determinado. Caso os dados de entrada ocupem um espaço maior do que o bloco em questão, há uma separação desses dados em pedaços correspondentes ao tamanho do bloco.

Este trabalho só lida com cifras de bloco e, portanto, segue a seguir na seção 2.1.2 uma explicação mais detalhada sobre o tema.

2.1.2 Cifras de bloco

Como descrito anteriormente, as cifras de blocos são algoritmos de encriptação realizados sobre blocos de dados que representam a mensagem a ser criptografada.

Parte do princípio de que a mensagem é dividida em vários blocos de dados, de tamanho pré-determinado, e o algoritmo de cifração converte cada bloco de dados num bloco encriptado.

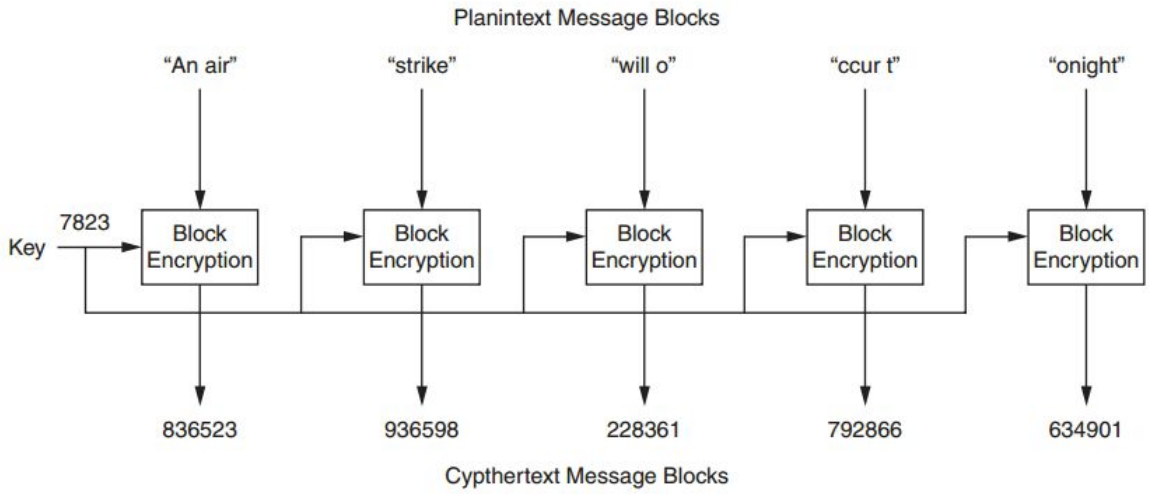
Como as cifras de bloco operam somente sobre blocos de tamanho fixo, é necessário tratar dois tipos de eventos: quando a mensagem de entrada é menor do que o tamanho do bloco, e quando a mensagem de entrada é maior do que o tamanho do bloco.

Para o primeiro caso (entrada menor do que o bloco), costuma-se utilizar a técnica de *padding*, que nada mais é do que o preenchimento da mensagem de entrada com bits suficientes até atingir o tamanho pré-determinado do bloco.

Para o segundo caso (entrada maior do que o bloco), aplicam-se técnicas conhecidas como modos de operação. Basicamente, a mensagem de entrada é dividida em vários blocos com o tamanho fixo da cifra, e o último bloco comumente possui tamanho diferente dos demais. Para este, aplica-se a técnica do *padding*, ou outra solução mais complexa.

Por exemplo, a Figura 2.3 abaixo demonstra o modo de operação mais simples existente, o ECB (*Electronic Code Book*).

Figura 2.3 - Cifra de bloco ECB



Fonte: [6], p. 28.

Esse modo de operação funciona da seguinte forma: cada bloco da mensagem original é transformado, via algoritmo e chave de encriptação, num bloco encriptado. Neste caso, o resultado da encriptação depende apenas do bloco de entrada e da chave utilizada, sem relação com os blocos vizinhos.

A fragilidade deste modo de operação é justamente o fato de que um mesmo bloco de entrada resulta na mesma saída, sempre. Os modos de operação mais robustos não possuem essa falha e, portanto, são mais seguros.

Para mais detalhes sobre o modo de operação utilizado neste trabalho, veja a seção 2.2.10. É denominado XTS, e resolve essa falha ao aplicar uma operação de "*xor*" entre os blocos vizinhos, além de operações de multiplicação polinomial binária.

2.2 TrueCrypt

O TrueCrypt é um software gratuito e de código aberto que proporciona *on-the-fly encryption* e *deniable encryption*. Pode ser utilizado para criar volumes virtuais de disco e montá-los na árvore de arquivo do seu computador, como se fosse um disco

real. Também permite criptografar uma partição, um dispositivo de armazenamento móvel, ou até o disco rígido inteiro. A Figura 2.4 abaixo apresenta o logotipo do TrueCrypt.

Figura 2.4 - Logotipo do TrueCrypt

Fonte: disponível em <http://www.legitreviews.com/wp-content/uploads/2014/05/truecrypt-logo.jpg>. Acessado em 15/08/2015.

O desenvolvimento do TrueCrypt foi descontinuado em 28 de maio de 2014 [1]. Após o evento, várias ramificações independentes do projeto foram feitas. A razão oficial para o encerramento do projeto é de que o programa não era suficientemente seguro [1], embora audições realizadas sobre o código fonte não demonstraram nenhum tipo de falha crítica que comprometa a integridade e a segurança dos dados armazenados. Para este trabalho, a versão utilizada como referência foi a 7.1a, que é a última versão com usabilidade completa disponível.

Nas subseções seguintes são apresentados alguns conceitos citados e informações relevantes referentes ao modo de funcionamento do TrueCrypt. Também são apresentadas algumas características e funcionalidades presentes no programa do TrueCrypt. Nem todas elas são implementadas neste trabalho, algumas por estarem fora do escopo e outras por não serem necessárias para atender às nossas necessidades, mas são apresentadas de forma a contextualizar as capacidades do programa.

Para informações referentes à utilização do TrueCrypt no computador pessoal, veja o APÊNDICE A - Utilização do TrueCrypt no computador pessoal.

2.2.1 On-the-fly encryption

O conceito de *on-the-fly encryption* representa uma maneira de criptografar os dados. Significa que os dados são automaticamente encriptados logo antes de serem salvos, e são decriptados logo antes de serem lidos, sem nenhum tipo de interferência do usuário no processo. Ou seja, é uma criptografia em tempo real, sem vestígios de dados armazenados de forma não segura.

2.2.2 Deniable encryption

O conceito de *deniable encryption* (criptografia negável) representa um conjunto de técnicas de criptografia que permite negar a existência de dados cifrados de forma convincente. Ou seja, um atacante que obtiver os dados cifrados não saberá distinguir se eles realmente existem ou se são um conjunto de dados aleatórios.

Parte do princípio da negação plausível (*plausible deniability*), que absolve o usuário da responsabilidades dos dados cifrados, uma vez que não é possível provar sua existência.

A implementação do *deniable encryption* no TrueCrypt se dá de três formas:

1. Volume oculto (ver seção 2.2.3);
2. Sistema operacional oculto (ver seção 2.2.4);
3. Partição ou dispositivo encriptado via TrueCrypt.

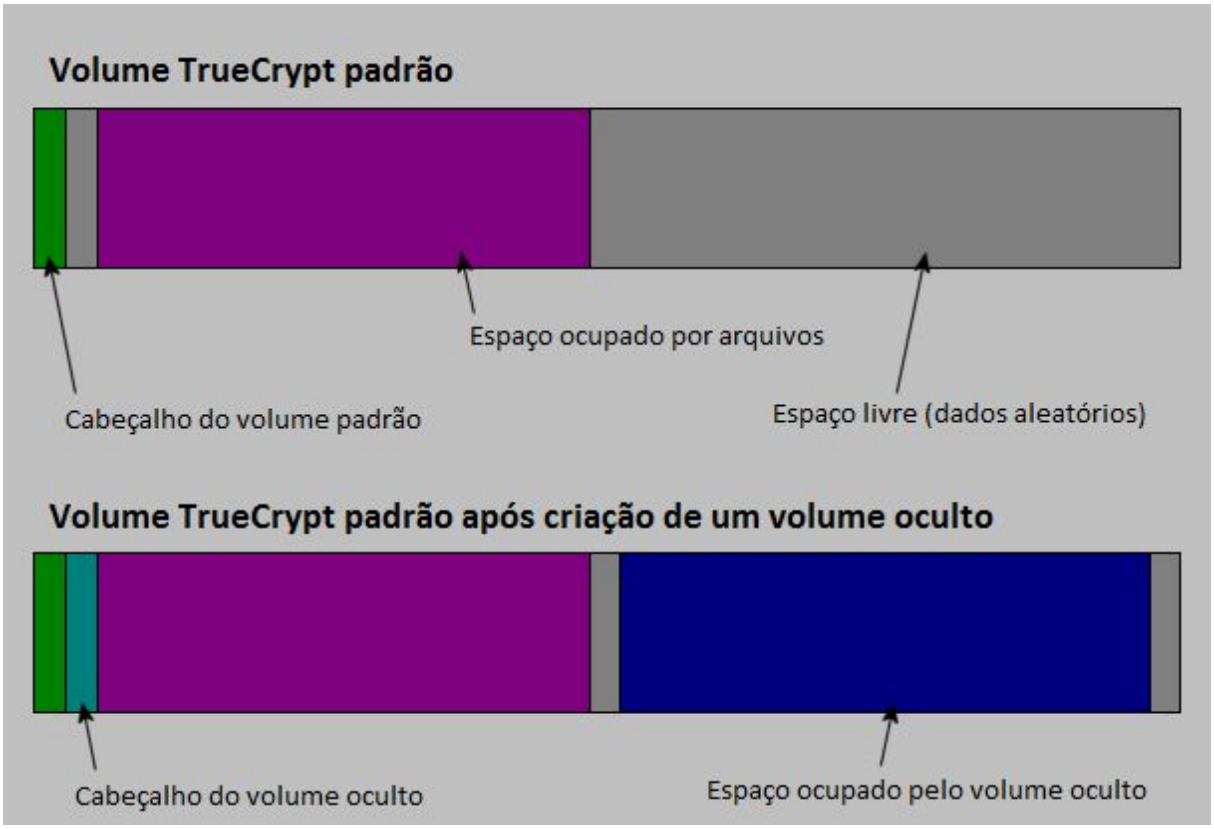
É importante destacar que os arquivos de volume TrueCrypt, embora pareçam ser apenas dados aleatórios depois de criptografados, não praticam *deniable encryption*

pois é injustificável a existência de arquivos contendo dados aleatórios sem nenhum motivo aparente.

2.2.3 Volume oculto

Um volume oculto nada mais é do que um volume de arquivos TrueCrypt dentro de outro volume de arquivos TrueCrypt. Quando um deles é montado, é impossível identificar a existência do outro sem o conhecimento da senha correta. Um exemplo de alocação de espaços desses volumes se dá na Figura 2.5 abaixo:

Figura 2.5 - Volume oculto



Fonte: Adaptado de [3].

A senha de decriptação deve ser diferente para os volumes. É a partir dela que o processo de decriptação reconhecerá se deve agir sobre o volume padrão ou o volume oculto. O volume montado será aquele cuja senha foi fornecida.

Para fortalecer o princípio de negação plausível, recomenda-se que o volume padrão seja preenchido com alguns arquivos “fantoches”, para despistar o possível invasor caso seja necessário divulgar uma senha (por motivos anteriormente já citados).

Uma particularidade do volume oculto é que seu tamanho é variável e dependente do tamanho total e do espaço ocupado pelo volume padrão. Dessa forma, deve-se ser cauteloso ao adicionar arquivos no volume padrão, pois é possível corromper a integridade dos arquivos no volume oculto caso haja invasão no seu espaço de memória.

Para detalhes da alocação de espaço de um volume TrueCrypt, veja a seção 2.2.11.

2.2.4 Sistema operacional oculto

Conforme citado na seção 2.2, o TrueCrypt é capaz de encriptar uma partição ou o próprio disco inteiro. Dessa forma, para o sistema operacional funcionar, é necessário uma senha de autenticação pré-*boot* para que o disco seja decriptado e o sistema operacional carregado na memória.

Seguindo o mesmo princípio do volume oculto, o TrueCrypt também permite que seja criado um sistema operacional oculto. São necessárias três senhas, uma para o sistema operacional “fantoche”, uma para o volume externo, e outra para o volume oculto com o sistema operacional oculto. A Figura 2.6 abaixo demonstra esse layout:

Figura 2.6 - Layout com sistema operacional oculto

Fonte: Adaptado de [3].

Para o pré-*boot*, entretanto, somente duas senhas serão utilizadas (do sistema operacional “fantoche” e do sistema operacional oculto). O *Boot Loader* do TrueCrypt identificará qual sistema operacional deve decriptar de acordo com a senha fornecida.

Como este trabalho não tratará da função de sistema operacional oculto, maiores detalhes foram omitidos. O sistema de *Boot Loader* do TrueCrypt e a função de encriptação de sistema operacional não serão utilizados nem referenciados novamente.

2.2.5 Paralelização

No caso do processador utilizado pelo computador possuir múltiplos núcleos, o TrueCrypt é capaz de paralelizar suas rotinas de encriptação e decriptação de forma que cada núcleo fique responsável por uma parcela dos dados envolvidos. Isso é feito dividindo-se um bloco de dados em pequenas partes, distribuídos entre cada núcleo. O resultado final é a reunião dos resultados parciais.

Dessa forma, a velocidade de encriptação e decriptação torna-se proporcional à quantidade de núcleos do processador em questão.

2.2.6 Pipelining

O TrueCrypt usa a estratégia de *pipelining* (processamento assíncrono) para encriptar e decriptar os dados.

Por exemplo, enquanto uma aplicação está carregando um arquivo que está encriptado, o TrueCrypt está automaticamente decriptando-o. Dessa forma, a aplicação não tem que esperar a finalização do processo de decriptação primeiro, ela pode carregar novas porções do arquivo diretamente.

2.2.7 Aceleração de hardware

O TrueCrypt utiliza aceleração de hardware por padrão caso seja identificado um processador com instruções AES disponíveis. Essas instruções estão presentes em alguns processadores Intel e AMD, e otimizam a velocidade de encriptação e decriptação utilizando o algoritmo AES (para maiores informações a respeito desse algoritmo, veja a seção 2.2.8).

2.2.8 Algoritmos de encriptação

A Tabela 2.1 abaixo lista os algoritmos de encriptação disponíveis no TrueCrypt:

Tabela 2.1 - Algoritmos de encriptação disponíveis no TrueCrypt

Algoritmo	Tamanho da chave (bits)	Tamanho do bloco (bits)	Modo de operação
AES	256	128	XTS
Serpent	256	128	XTS
Twofish	256	128	XTS
AES-Twofish	256; 256	128	XTS
AES-Twofish-Serpent	256; 256; 256	128	XTS
Serpent-AES	256; 256	128	XTS
Serpent-Twofish-AES	256; 256; 256	128	XTS
Twofish-Serpent	256; 256	128	XTS

Fonte: Adaptado de [3].

A seguir, são apresentadas as características básicas de cada um desses algoritmos. Quanto ao modo de operação XTS, veja a seção 2.2.10.

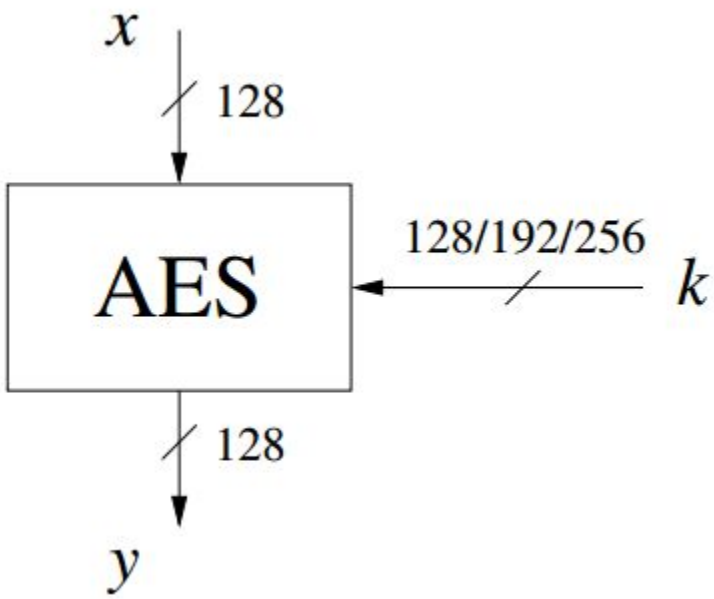
Os algoritmos combinados são apenas encriptações em cascata dos blocos de dados.

2.2.8.1 AES

O *Advanced Encryption Standard* (AES) é o algoritmo de chave simétrica mais amplamente utilizado atualmente. Sua especificação foi adotada pelo governo americano a partir de uma tentativa de padronização para seus sistemas de segurança. Atualmente, é utilizado em diversas aplicações comerciais.

É baseado na cifra Rijndael. Possui um bloco de tamanho 128 bits e chaves de tamanhos 128,192 e 256 bits. A Figura 2.7 representa essa configuração:

Figura 2.7 - AES



Fonte: [5], p. 89.

Para maiores detalhes, ver [11].

2.2.8.2 Serpent

O algoritmo *Serpent* foi um dos finalistas no concurso para definição do AES, mas ficou em segundo lugar em relação ao Rijndael. Também possui um bloco de tamanho 128 bits e chaves de tamanhos 128,192 e 256 bits. Considera-se que possui uma margem de segurança maior do que o Rijndael [14], porém isso reduziria a eficiência do algoritmo em implementações de software.

Para maiores detalhes, ver [12].

2.2.8.3 Twofish

O algoritmo *Serpent* ficou entre os cinco finalistas no concurso para definição do AES. Também possui um bloco de tamanho 128 bits e chaves de tamanhos 128,192

e 256 bits. Uma particularidade sua é a utilização de *S-boxes* dependentes de chaves e alta complexidade nas rodadas de cifração.

Para maiores detalhes, ver [13].

2.2.9 Funções de hash

As funções de *hash* são utilizadas para se obter as chaves de encriptação do cabeçalho (ver seção 2.2.11). Estas funções recebem uma cadeia de bits de tamanho arbitrário e realizam o mapeamento para uma cadeia de tamanho fixo.

O TrueCrypt disponibiliza três funções distintas: *RIPEMD-160*, *SHA-512* e *Whirpool*. Nos três casos, o tamanho da cadeia de saída é de 512 bits.

A seguir, são apresentadas as características básicas de cada uma dessas funções.

2.2.9.1 RIPEMD-160

O RIPEMD-160 é uma função de *hash* de 160 bits, desenvolvido para ser uma melhoria ao RIPEMD original. É uma solução mais difícil de ser quebrada e, portanto, mais segura do que sua antecessora.

Para maiores detalhes, ver [15].

2.2.9.2 SHA-512

O SHA-512 é uma função de *hash* de 512 bits, desenvolvida pela NSA (*National Security Agency*). É uma evolução do SHA-1, que é uma função de 160 bits e que já é considerado fraco e pouco seguro frente às tecnologias atuais [16].

Para maiores detalhes, ver [16] e [17].

2.2.9.3 Whirpool

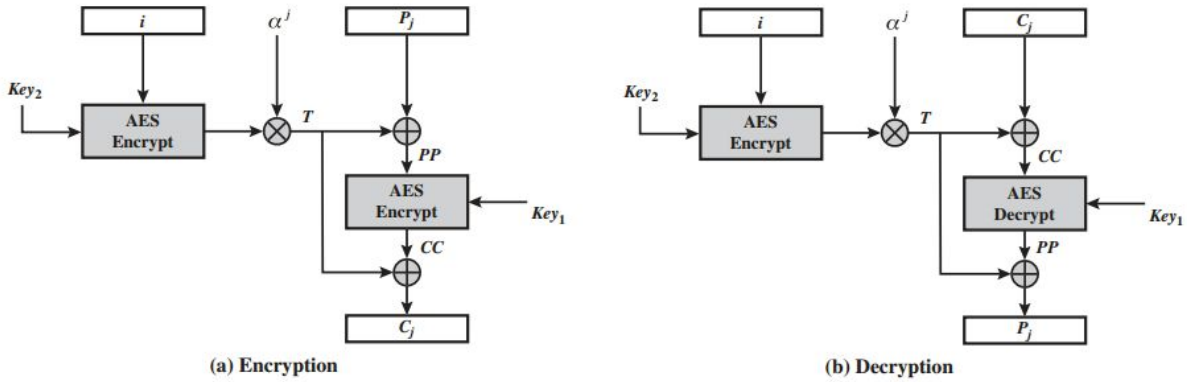
O *Whirpool* é uma função de *hash* de 512 bits, adotado pela ISO (*International Organization for Standardization*) e pela IEC (*International Electrotechnical Commission*). Aplica o método de *hashing* Miyaguchi-Preneel e é baseado nos princípio de cifração de bloco do AES.

Para maiores detalhes, ver [18].

2.2.10 XTS

O XTS é um modo de operação aprovado pelo IEEE para cifração de blocos. É uma alteração do XEX (*Xor-Encrypt-Xor*) com alguns ajustes e adição de *ciphertext-stealing* (CPS). Seu funcionamento é representado na Figura 2.8 abaixo:

Figura 2.8 - Operação do XTS em um bloco de dados



Fonte: [4], p.195.

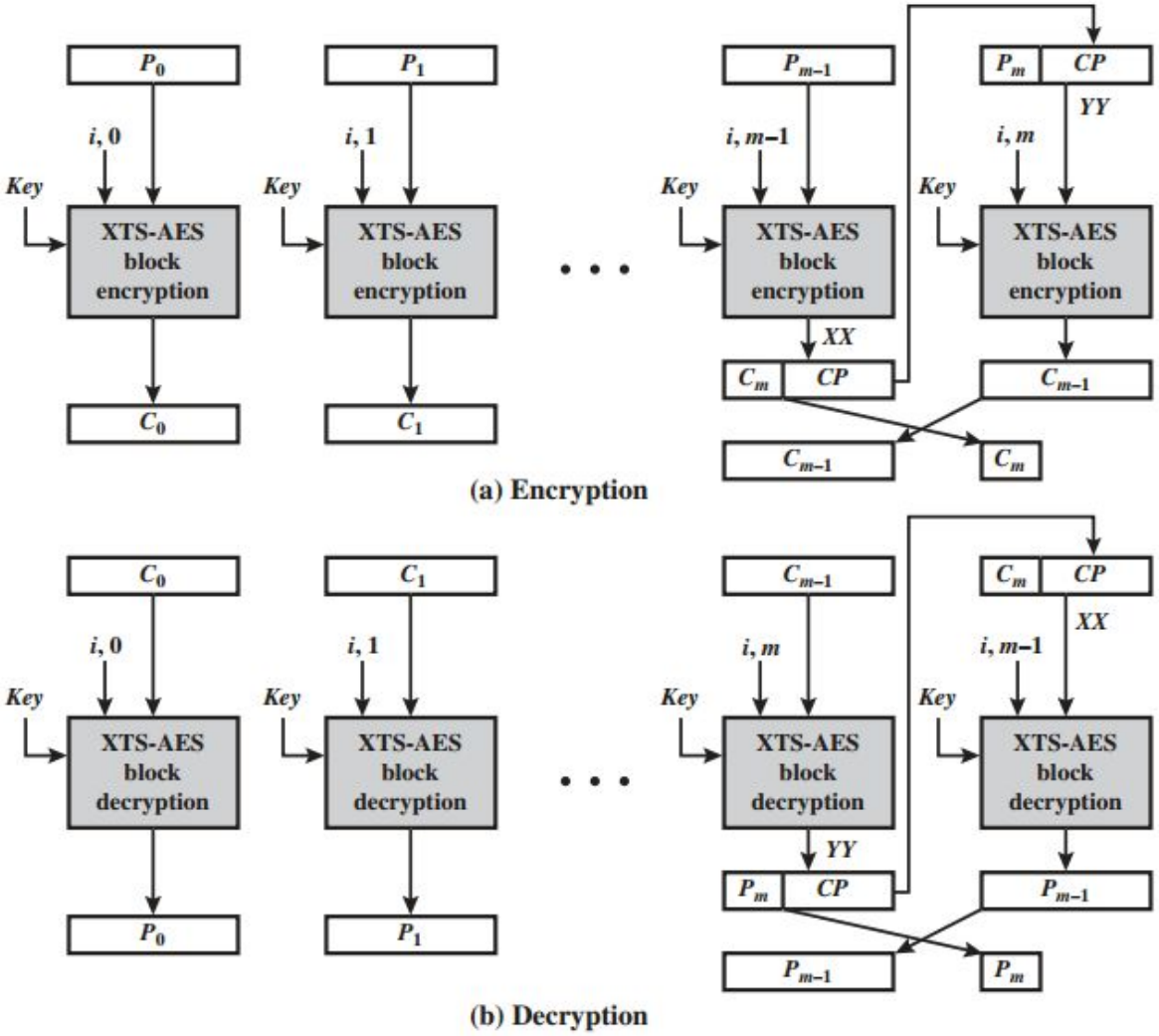
Os parâmetros envolvidos são:

- Key chave de 256 bits do(s) algoritmo(s) de encriptação utilizado(s);
- Pj o *j*º bloco de dado. Todos os blocos exceto o último possuem 128 bits;
- Cj o *j*º bloco cifrado. Todos os blocos exceto o último possuem 128 bits;
- j número sequencial do bloco dentro da unidade de dado;
- i ajuste de 128 bits, determinado sequencialmente a partir do primeiro bloco como um valor inteiro não negativo, iniciado por um valor arbitrário;

- α elemento primitivo do *Galois Field* (2^{128}) - campo finito - que corresponde ao polinômio *x* (ou seja, $0000...010_2$);
- α^j α multiplicado por si mesmo j vezes, em GF(2^{128});
- $\oplus$ XOR
- $\otimes$ multiplicação dos polinômios binários com módulo $x^{128}+x^7+x^2+x+1$, em GF(2^{128}).

No caso do último bloco não possuir 128 bits, utiliza-se o método *ciphertext-stealing*, que mantém o tamanho do bloco de dado após a cifração. Esse método é representado na Figura 2.9 abaixo:

Figura 2.9 - *Ciphertext-stealing* no XTS



Fonte: [4], p. 197.

No TrueCrypt, cada unidade de dado é de 512 bytes, independente do tamanho do setor no dispositivo de armazenamento. Portanto, como cada bloco básico do XTS é de 128 bits, cada unidade de dado do TrueCrypt precisa passar por 32 sequências de blocos XTS.

Para maiores detalhes sobre o XTS, ver [19].

2.2.11 Especificação do volume TrueCrypt

Um volume TrueCrypt possui uma especificação padrão para a posição de informações necessárias para a sua decriptação. Os primeiros 128 KB são considerados o cabeçalho (*header*) do volume e contém informações como versão, *checksum*, tamanho do volume, *salt*, entre outros. A Tabela 2.1 abaixo especifica a posição dos bytes, o tamanho e a informação correspondente existentes no volume.

Tabela 2.2 - Especificação do formato de volume TrueCrypt

Offset (bytes)	Tamanho (bytes)	Descrição
0	64	Salt
64	4	ASCII string "TRUE"
68	2	Volume header format version (5)
70	2	Minimum program version required to open the volume
72	4	CRC-32 checksum of the (decrypted) bytes 256–511
76	16	Reserved (must contain zeroes)
92	8	Size of hidden volume (set to zero in non-hidden volumes)
100	8	Size of volume
108	8	Byte offset of the start of the master key scope
116	8	Size of the encrypted area within the master key scope
124	4	Flag bits (bit 0 set: system encryption; bit 1 set: non-system in-place-encrypted volume; bits 2-31 are reserved)
128	4	Sector size (in bytes)
132	120	Reserved (must contain zeroes)
252	4	CRC-32 checksum of the (decrypted) bytes 64–251
256	Var.	Concatenated primary and secondary master keys
512	65024	Reserved
65536	65536	Area for hidden volume header (if there is no hidden volume within the volume, this area contains random data**).
131072	Var.	Data area (master key scope). For system encryption, offset may be different (depending on offset of system partition).
S–131072	65536	Backup header (encrypted with a different header key derived using a different salt).
S–65536	65536	Backup header for hidden volume (encrypted with a different header key derived using a different salt). If there is no hidden volume within the volume, this area contains random data

Fonte: Adaptado de [3].

Como pode-se notar, dentro desses 128 KB de cabeçalho, 64 KB são de informações do volume principal, e os próximos 64 KB são de informações do volume oculto, se ele existir. Condiz com a Figura 2.1 da seção 2.2.3.

É importante observar que desses 64 KB de espaço reservado para os cabeçalhos, apenas os primeiros 512 bytes possuem informação relevante.

Após os 128 KB, temos a área de dados do volume. No final, os últimos 128 KB também são de cabeçalho (de *backup*). Deles, podem ser obtidos as mesmas informações do cabeçalho principal. A diferença é que estão encriptados de formas distintas e, portanto, só são convenientes para utilização no caso do cabeçalho principal estar corrompido.

2.2.12 Criação de um volume

A seguir, uma descrição sucinta do processo de criação de um volume do TrueCrypt:

1. Inicialmente, cada setor do novo volume é formatado. Isso significa que o novo volume é inteiramente preenchido com dados aleatórios;
2. Utiliza-se a senha fornecida pelo usuário, a função de *hash* escolhida e o *salt* gerado aleatoriamente para gerar a chave do cabeçalho, após um processo de fortalecimento de chave (*key strengthening*);
3. As chaves para a área de dados são geradas de forma aleatória e inseridas no cabeçalho;
4. O cabeçalho é encriptado utilizando o(s) algoritmo(s) de encriptação escolhido(s) e sua respectiva chave;
5. A área de dados é encriptada utilizando o(s) algoritmo(s) de encriptação escolhido(s) e suas respectivas chaves;

2.2.13 Decriptação

A decriptação de um volume ocorre de forma análoga à encriptação. A seguir, uma descrição sucinta do processo:

1. Dos primeiros 512 bytes do volume, extrai-se o *salt* (primeiros 64 bytes) e determina-se como cabeçalho efetivo os bytes restantes (os próximos 448 bytes);
2. Como não sabemos a função de *hash* e o algoritmo de encriptação utilizados, é necessário testar todas as combinações possíveis. Itera-se uma tentativa de decriptação do header usando as combinações formadas;
3. A combinação é válida quando a decriptação do header forma a palavra “TRUE” em ASCII nos bytes iniciais do cabeçalho;
4. No header decriptado também estão contidas as chaves para decriptação da área de dados do volume. Utilizam-se essas chaves em conjunto com o algoritmo de encriptação, agora conhecido, para decriptar o volume completo.
5. Caso nenhuma das combinações da etapa 2 funcione (ou seja, nenhuma retorna a palavra “TRUE” no cabeçalho), tenta-se novamente o processo sobre o cabeçalho do volume oculto (512 bytes a partir do byte 65536).

É importante notar que a senha fornecida pelo usuário é somente a senha de decriptação do cabeçalho do volume, e não da área de dados. A senha da área de dados está contida dentro do cabeçalho.

2.3 Sistema de arquivos

Um sistema de arquivos é um sistema que controla a forma como os dados são armazenados e recuperados de um dispositivo de armazenamento, como por exemplo disco rígidos, memórias *flash*, fitas magnéticas, etc.

O sistema é responsável por diversas funções, sendo as principais:

- Organizar o espaço de memória;
- Identificar a localização dos dados;
- Manter a integridade dos dados;

- Estabelecer uma hierarquia de diretórios;
- Armazenar metadados de arquivos (ex: tamanho, atributos, data de criação e modificação, etc.);
- Controlar permissões de acesso.

Existem vários tipos de sistema de arquivos. Alguns são específicos para determinado tipo de dispositivo de armazenamento (por exemplo ISO 9660 para mídias de disco óptico), enquanto outros são genéricos e funcionam em vários tipos de dispositivos (por exemplo NTFS e FAT).

Este trabalho lida com o sistema de aquivos FAT e, portanto, segue a seguir na seção 2.3.1 uma descrição mais detalhada sobre ele.

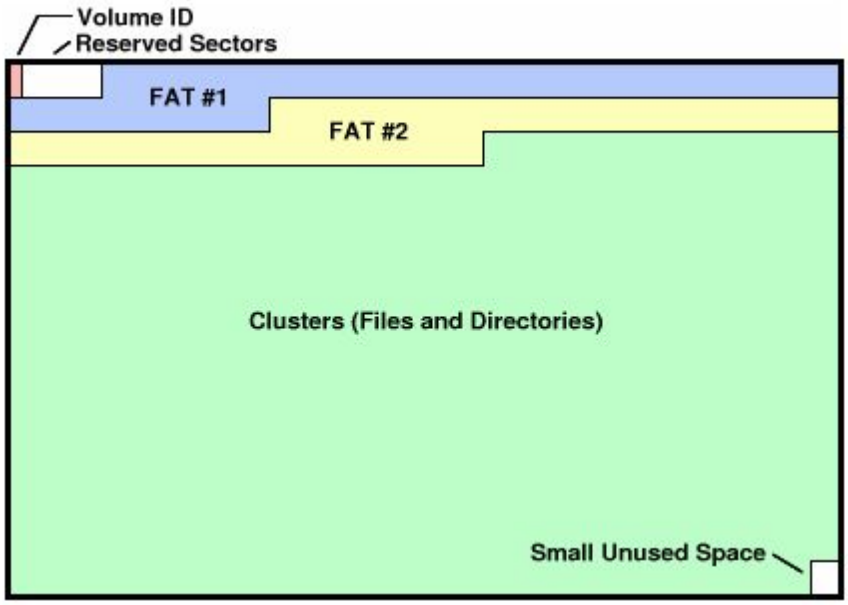
2.3.1 FAT

O FAT (*File Allocation Table*) é um sistema de arquivos atualmente utilizado em grande escala por diversos tipos dispositivos. Foi projetado originalmente para funcionar com disquetes (de mídia magnética) e foi adaptado e utilizado na era Windows 9x e DOS.

Sua história conta com três variantes importantes: FAT12, FAT16 e FAT32. Devido ao estado da tecnologia de armazenamento atual, apenas o FAT32 ainda possui utilização em grande escala. As variantes mais antigas sofrem de limitações quanto ao espaço de partição e tamanho máximo de arquivo suportado.

O FAT32 organiza-se da forma como mostra a Figura 2.10 abaixo:

Figura 2.10 - FAT32

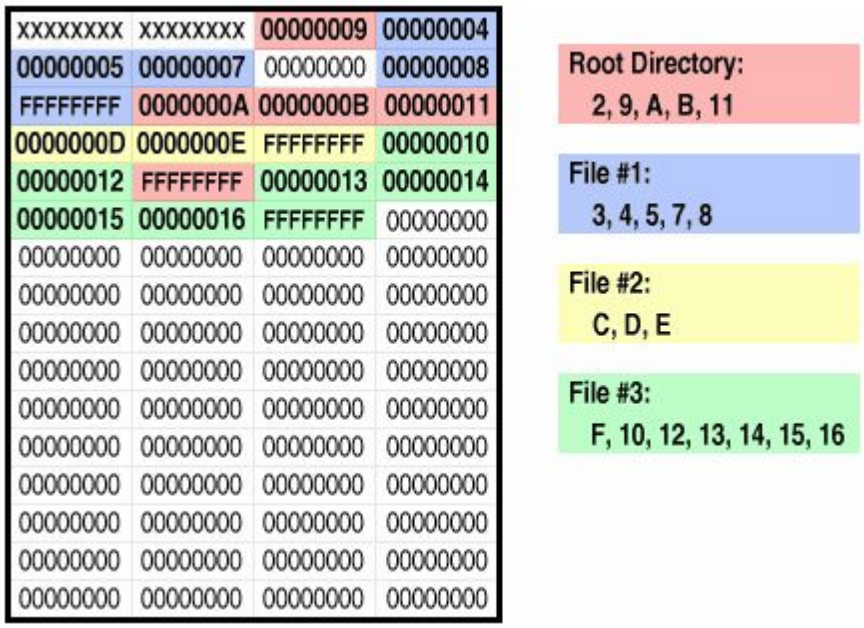


Fonte: [21]

O primeiro setor do volume consiste no MBR (*Master Boot Record*) e consiste em código de *boot* do volume. Depois, existe o setor chamado de Volume ID e contém informações sobre o próprio volume em questão (por exemplo o número de bytes por setor, setores por *cluster*, etc.). Após alguns setores reservados, estão presentes as FATs propriamente ditas, que nada mais são do que estruturas de dados indicativas da localização dos arquivos e diretórios nos *clusters*.

O armazenamento físico dos dados não é necessariamente linear e pedaços de um arquivo podem estar espalhados dentro de um setor. Para localizar um arquivo e reconstituí-lo, as FATs armazenam ponteiros para as posições de memórias que constituem a cadeia de *clusters* a serem acessados. A Figura 2.11 abaixo ilustra esse caso.

Figura 2.11 - Setor e cadeia de clusters



Fonte: [21]

O FAT32 possui um contador de 32 bits para os setores, o que faz com que o tamanho total de um volume FAT seja limitado de 2 TB (no caso de tamanho de setor de 512 bytes) a 16 TB (no caso de tamanho de setor de 4 KB). O tamanho máximo de um arquivo dentro de um sistema FAT32 é de 4 GB menos 1 byte ou 4.294.967.295 ($2^{32} - 1$) bytes.

Para mais detalhes, ver [20] e [21].

3 ESPECIFICAÇÃO

Este capítulo tem como objetivo apresentar uma especificação concisa do aplicativo a ser desenvolvido. Para isso, serão descritos os requisitos que o sistema deve apresentar, os casos de usos identificados e eventuais diagramas e modelos necessários para um melhor entendimento do mesmo.

3.1 Especificação de requisitos

3.1.1 Requisitos funcionais

1. Compatibilidade com Android versões 5.0 ou superiores;
2. Compatibilidade com o TrueCrypt: o aplicativo deve criar e montar volumes do formato TrueCrypt, respeitando a posição dos dados, a estrutura do cabeçalho e as funções criptográficas de encriptação e decriptação;
3. Volumes no formato FAT32: o sistema de arquivos correspondente dos volumes trabalhados pelo aplicativo devem ser desse formato especificado;
4. Navegação no sistema de arquivos: o aplicativo deverá ser capaz de navegar dentro da árvore de arquivos do sistema;
5. Importar arquivos: o aplicativo deve permitir a importação de arquivos já existentes no dispositivo móvel para dentro de um volume TrueCrypt.
6. Exportar arquivos: o aplicativo deve permitir a exportação de arquivos de dentro de um volume TrueCrypt para o dispositivo móvel.

3.1.2 Requisitos não-funcionais

1. Segurança: as senhas criptográficas utilizadas pelo usuário jamais serão salvas na memória, e os dados dos arquivos importados e exportados nunca serão manipulados de forma vulnerável;
2. Usabilidade: o aplicativo deverá possuir uma interface clara e intuitiva, de forma que o usuário não encontre dificuldades no manuseio das suas funções;
3. Desempenho: o aplicativo deverá ser capaz de realizar as funções de encriptação e decriptação num tempo limite de um minuto.

3.2 Casos de uso

Nesta seção encontram-se os casos de uso identificados do aplicativo. Serão descritos seguindo o seguinte modelo: descrição, evento iniciador, pré-condição, sequência de eventos, pós condição, extensões e inclusões.

3.2.1 Criar Volume

Descrição: Este caso descreve o processo de criação de um novo volume.

Evento iniciador: Clique no botão “Novo volume” na tela inicial.

Pré-condição: nenhuma.

Sequência de eventos:

1. Usuário seleciona a opção “Novo volume” na tela inicial.

2. Sistema solicita os parâmetros do novo volume: local a ser salvo (*path*), tamanho, função de *hash* e algoritmo de encriptação.

3. Usuário fornece os parâmetros desejados.

4. Sistema solicita a senha.

5. Usuário fornece a senha.

6. Sistema cria o volume e informa ao usuário o sucesso da operação.

Pós-condição: volume salvo no disco.

Extensões:

1. Espaço em disco insuficiente: sistema exibe uma mensagem ao usuário e retorna à tela inicial sem criar o volume (passo 6).

3.2.2 Montar Volume

Descrição: Este caso descreve o processo de montagem de um volume já criado.

Evento iniciador: Clique no botão “Montar volume” na tela inicial.

Pré-condição: volume salvo no disco.

Sequência de eventos:

1. Usuário seleciona a opção “Montar volume” na tela inicial.

2. Sistema solicita o volume a ser montado.

3. Usuário seleciona o arquivo do volume.

4. Sistema solicita a senha.

5. Usuário fornece a senha.

6. Sistema decripta o header do volume e o monta utilizando o módulo FUSE.

7. Sistema exibe os arquivos e diretórios na raíz do volume.

Pós-condição: volume montado no local padrão.

Extensões:

1. Senha incorreta: sistema informa o usuário do erro e retorna à tela inicial (passo 6).

Inclusões:

1. Sistema cadastra o volume no módulo FUSE em um local padrão.

3.2.3 Desmontar Volume

Descrição: Este caso descreve o processo de desmontagem de um volume.

Evento iniciador: Clique no botão “Desmontar volume” na tela inicial.

Pré-condição: volume montado.

Sequência de eventos:

1. Usuário seleciona a opção “Desmontar volume” na tela inicial.

2. Sistema lista os volumes montados.

3. Usuário seleciona o volume.

4. Sistema solicita ao módulo FUSE que desmonte o volume e informa ao usuário que o mesmo foi desmontado.

Pós-condição: volume desmontado.

Extensões:

1. Volume em uso: sistema informa ao usuário que não foi possível desmontar o volume e retorna à tela principal (passo 4).

3.2.4 Abrir um arquivo dentro do volume

Descrição: Este caso descreve o processo de abertura de um arquivo existente dentro de um volume.

Evento iniciador: Usuário seleciona um arquivo dentro do volume.

Pré-condição: volume montado.

Sequência de eventos:

1. Usuário seleciona um arquivo.

2. Sistema lista os aplicativos disponíveis de acordo com o tipo de arquivo.

3. Usuário seleciona o aplicativo.

4. Sistema chama o aplicativo selecionado fornecendo o local do arquivo.

Pós-condição: nenhuma.

Extensões:

1. Tipo de arquivo desconhecido: sistema informa a inexistência de um aplicativo capaz de abrir o arquivo selecionado e volta a tela de seleção de arquivo (passo 1).

3.2.4 Exportar arquivo

Descrição: Este caso descreve o processo exportação de um arquivo para fora do volume criptografado.

Evento iniciador: Usuário seleciona um arquivo dentro do volume.

Pré-condição: volume montado.

Sequência de eventos:

1. Usuário seleciona um arquivo.

2. Sistema exibe uma mensagem de alerta informando os riscos de segurança da operação e solicita confirmação.

3. Usuário confirma a operação.

4. Sistema solicita o caminho onde o arquivo deverá ser salvo.

5. Usuário fornece o caminho.

6. Sistema decripta o arquivo e o copia no local informado.

Pós-condição: nenhuma.

Extensões:

1. Espaço insuficiente: sistema informa falta de espaço no destino escolhido e retorna à lista de arquivos (passo 1).

3.2.4 Importar arquivo

Descrição: Este caso descreve o processo exportação de um arquivo para dentro do volume criptografado.

Evento iniciador: Usuário seleciona um arquivo dentro do volume.

Pré-condição: volume montado.

Sequência de eventos:

1. Sistema solicita o caminho do arquivo a ser importado.

2. Usuário fornece o caminho do arquivo.

2. Sistema criptografa o arquivo.

3. Sistema copia o arquivo para dentro do volume.

Pós-condição: arquivo contido no volume.

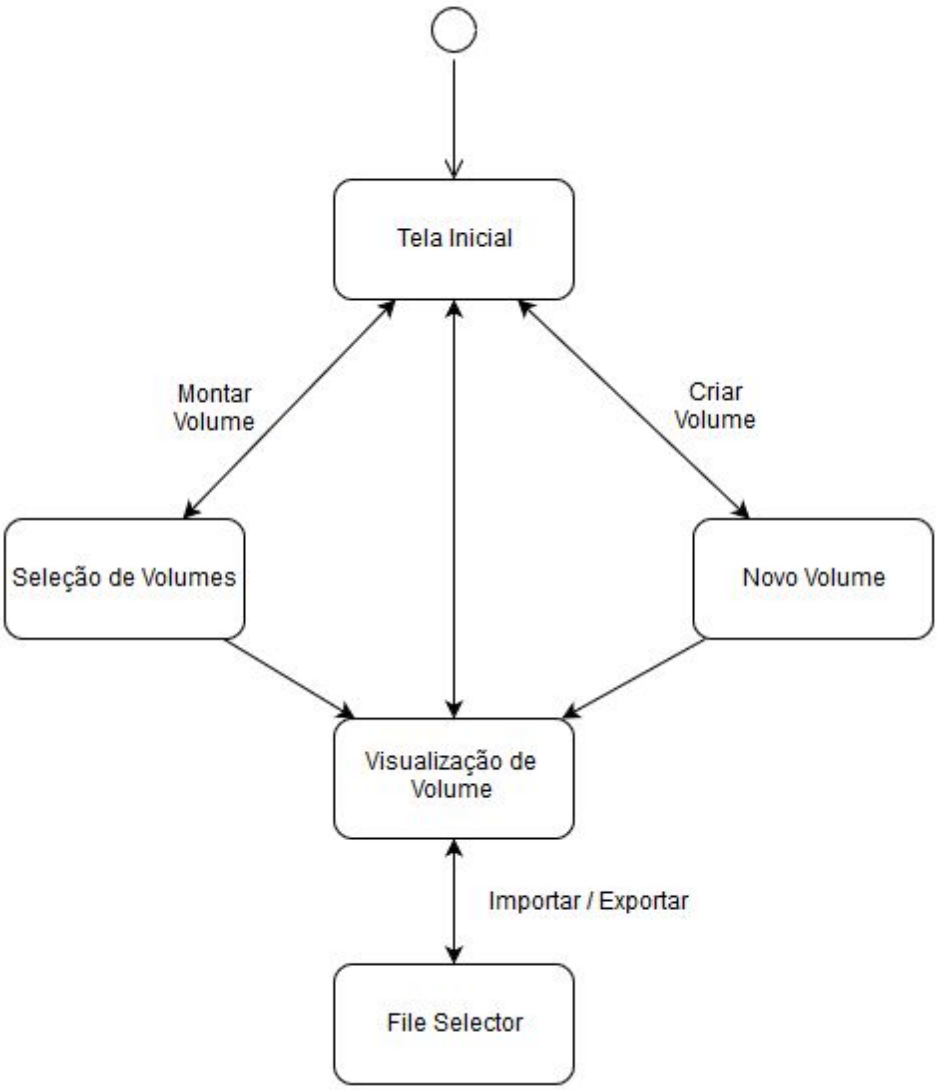
Extensões:

1. Espaço insuficiente: sistema informa falta de espaço no volume e retorna à lista de arquivos (passo 1).

3.3 Interface do aplicativo

A Figura 3.1 abaixo apresenta o modelo de navegação simplificado do sistema.

Figura 3.1 - Modelo de navegação



Fonte: elaborado pelos autores.

A seguir, para cada elemento, serão descritos sua função e sua interface visual no aplicativo.

3.3.1 Tela inicial

A Figura 3.2 demonstra um rascunho da tela inicial do aplicativo. Nela, constam apenas dois botões, um para criar um volume novo e outro para montar um volume já existente.

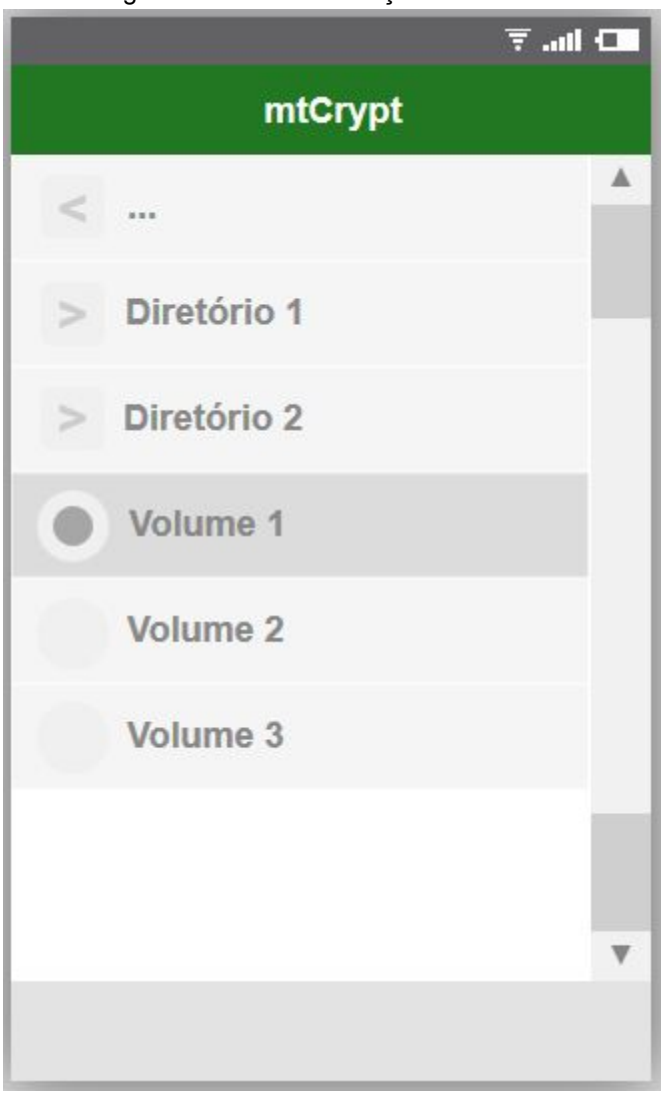
Figura 3.2 - Tela inicial

Fonte: elaborado pelos autores.

3.3.2 Seleção de volumes

A Figura 3.3 demonstra um rascunho da tela de seleção de volumes. Ela é basicamente um navegador de arquivos, no qual o usuário deverá selecionar um volume válido que será decriptado e montado no sistema de aquivos.

Figura 3.3 - Tela de seleção de volumes



Fonte: elaborado pelos autores.

3.3.3 Novo Volume

A Figura 3.4 demonstra um rascunho da tela de criação de um novo volume. Ela é um formulário que pede as informações para a encriptação do volume, como por exemplo função de *hash*, algoritmo de encriptação e senha.

Figura 3.4 - Tela de novo volume

Fonte: elaborado pelos autores.

3.3.4 Visualização de Volume e File Selector

A Figura 3.5 demonstra um rascunho da tela de visualização de um volume, após montado. Ela é basicamente um navegador de arquivos, tendo como raiz o caminho de montagem do volume no sistema de arquivos.

Figura 3.5 - Tela de visualização de volume

Fonte: elaborado pelos autores.

Nesta mesma interface também estão presentes as funções de importar e exportar um arquivo. Para importar, o usuário aperta o botão de “Importar arquivo” do canto superior esquerdo e uma caixa aparece requisitando o caminho do arquivo a ser

importado. Para exportar, o usuário aperta o botão de “Exportar arquivo” e uma caixa aparece requisitando o caminho para onde o arquivo será exportado.

4 IMPLEMENTAÇÃO

Esta seção descreve a forma de implementação da especificação proposta na seção 3. Procura descrever as tecnologias envolvidas, os métodos adotados, arquivos utilizados e os artefatos produzidos do trabalho.

4.1 Tecnologias utilizadas

A seguir, apresentamos as tecnologias utilizadas na implementação do mtCrypt. As referências são citadas nas subseções e recomenda-se a utilização das mesmas para maiores informações a respeito da tecnologia. A Figura 4.1 abaixo ilustra as tecnologias utilizadas:

Figura 4.1 - Tecnologias utilizadas

Fonte: elaborado pelos autores.

4.1.1 Android [24]

O Android é um sistema operacional aberto da Google™ para dispositivos móveis. Neste projeto será utilizado a versão 6.0 ("*Marshmallow*"). Diferentemente das versões normalmente presente nos dispositivos comercializados, será utilizado uma versão com permissão de acesso de superusuário ("*root"*) devido a alguns requisitos específicos deste projeto, que não são acessíveis com as permissões de usuário comum, mais especificamente a manipulação da árvore de arquivos do sistema.

4.1.2 Java [25]

O Java é uma linguagem orientada a objetos desenvolvida para ser independente da máquina onde é executada, utilizando, para isso, uma máquina virtual - a JVM. É a linguagem padrão para desenvolvimento de aplicativos para a plataforma android.

4.1.3 Android Studio [26]

O Android Studio é um IDE (*Integrated Development Environment*) projetado especificamente para desenvolvimento na plataforma Android, desenvolvida pela própria Google. É baseado no IntelliJ IDEA, que é um famoso IDE de Java, criada pela JetBrains.

4.1.4 Android SDK [27]

Um *software development kit* (SDK) é um pacote que permite um programador desenvolver aplicativos para uma plataforma específica. No caso do Android SDK, são fornecidas as bibliotecas da API e as ferramentas necessárias para o desenvolvedor criar, testar e depurar aplicativos para Android.

A versão do SDK utilizada para compilação do projeto é a 23, que condiz com a versão 6.0 (“*Marshmallow*”) do Android.

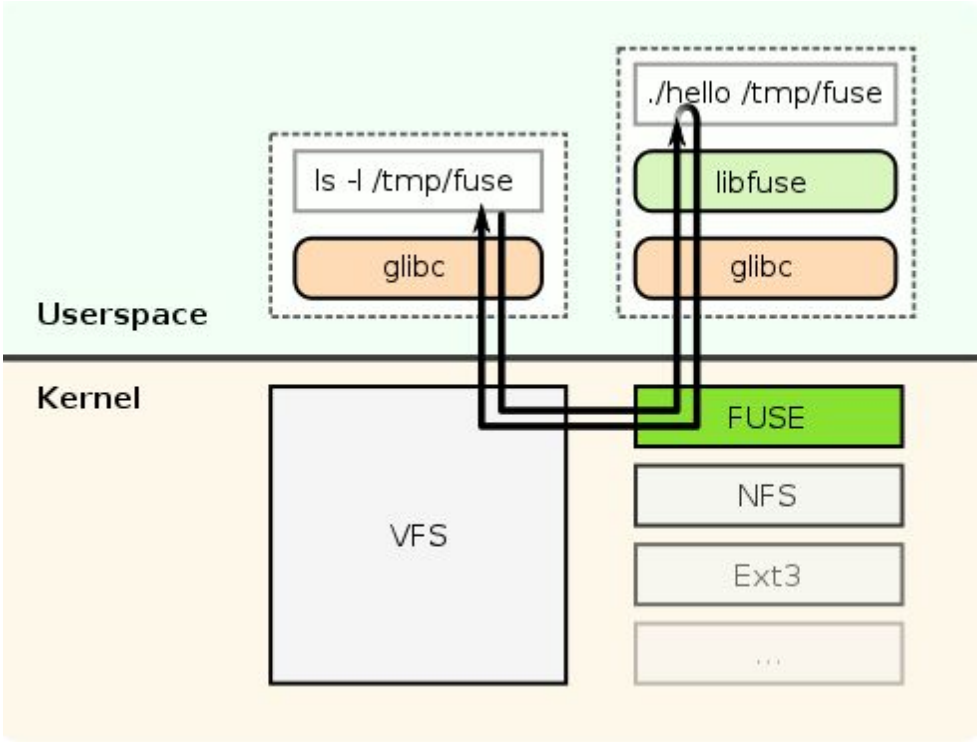
4.1.3 Gnu-crypto [28]

A gnu-crypto é uma biblioteca do projeto GNU disponível nas linguagens Java e C# que fornece diversas primitivas criptográficas. Todas as primitivas necessárias ao TrueCrypt (e por consequência ao mtCrypt) estão disponíveis na mesma. Consistem nos algoritmos de encriptação (seção 2.1.8) e nas funções de Hash (seção 2.1.9).

4.1.4 FUSE [29]

O *“Filesystem in Userspace”* (FUSE) é um modulo presente no *kernel* do Android que permite a criação e manipulação de sistemas de arquivos sem que seja necessário modificar o código fonte do *kernel*. O FUSE age como uma ponte entre a aplicação e o módulo do *kernel* responsável pelos sistemas de arquivos, como demonstrado na Figura 4.2 abaixo:

Figura 4.2 - FUSE



Fonte: [19].

Nesta imagem, a implementação de um programa “*hello*” faz com que a execução do comando “*ls*” seja realizada pelo FUSE. Dessa forma, existe uma intermediação do acesso ao sistema de arquivos. Isso também possibilita a alteração da função básica, como adição de logs, mensagens ao usuário, entre outros.

4.1.5 FAT32 Library [30]

A FAT32 Library (fat32-lib) é uma biblioteca desenvolvida por Matthias Treydte, que possibilita a criação e manipulação de sistemas de arquivo do tipo FAT12, FAT16 e FAT32. É completamente implementada em Java, não possui dependências externas e seu uso é livre.

4.1.6 Git [31] *e GitHub* [32]

O Git é um sistema de controle de versão de código livre e gratuito. Será utilizado para facilitar o desenvolvimento em paralelo da equipe.

O GitHub é uma plataforma de compartilhamento de código e será utilizado como servidor de repositórios Git, para armazenar nosso projeto.

4.2 Código desenvolvido

O código do aplicativo desenvolvido será explicado com base nas classes e funções implementadas. Não será apresentado o código completo do aplicativo desenvolvido, apenas sua estrutura de classes e relações entre as chamadas de funções.

MainActivity

Essa classe é padrão no desenvolvimento de aplicativos para Android e trata da interface e das funções utilizadas pela tela inicial do aplicativo, logo após sua abertura. Nessa classe estão presentes as funções de criar volume e abrir volume, e também faz a inicialização do serviço de volume que trata da manipulação de um volume de dados (explicado com maiores detalhes na classe *volumeService*).

volumeExplorerActivity

Essa classe trata da interface e das funções utilizadas pela tela de navegação de um volume. Representa a listagem dos arquivos e diretórios, navegação através da árvore de arquivos, seleção e manipulação de arquivos e diretórios e funções de

importar e exportar arquivos. É categorizada como uma classe de controle entre a interface da tela e as implementações das funções citadas acima, que são realizadas efetivamente pelo serviço de volume (explicado com maiores detalhes na classe *volumeService*).

CustomArrayAdapter

Essa é uma classe auxiliar à classe *volumeExplorerActivity*, e apenas estende a estrutura de dados *ArrayAdapter* que é utilizada na listagem de arquivos e diretórios na tela de navegação de um volume. Ela facilita a inserção de um ícone representativo ao lado do texto que nomeia um arquivo ou diretório, fazendo a distinção entre deles.

volumeService

Essa classe é responsável pela manipulação de um volume de dados e dos arquivos e diretórios contidos nele. Funções como navegação pela árvore de arquivos, criação e remoção de diretórios, remoção de arquivos, importação e exportação de arquivos e informações gerais como espaço livre e espaço total são implementadas nessa classe. Também é aqui que é realizada a encriptação e decriptação de um volume e, portanto, essa classe é responsável pela chamada das classes criptográficas.

tcVolume

Essa classe faz a representação de um volume TrueCrypt e é responsável pela decriptação de seu cabeçalho e extração das informações relevantes para a decriptação da área de volumes (que é realizado pela classe *tcBlockDevice*).

É instanciada a partir do arquivo de volume selecionado e da senha fornecida pelo usuário, e a decriptação do cabeçalho extrai informações como algoritmo de encriptação utilizado e as senhas para a decriptação da área de volumes. Também são realizadas as devidas validações de *checksum* para garantir a integridade dos dados decriptados.

tcBlockDevice

Essa classe é a implementação de um volume TrueCrypt considerando a área de dados, utilizando a interface *BlockDevice* para integração com a biblioteca *fat32-lib*. É instanciada após a decriptação do cabeçalho e, com posse de todas as informações necessárias, é responsável pela escrita e leitura do volume utilizando o modo XTS.

A implementação como interface *BlockDevice* e utilização das funções da biblioteca *fat32-lib* fazem com que o volume de dados decriptado seja tratado como um volume FAT32 e facilmente montado no sistema de arquivos do dispositivo móvel.

Xts

Essa classe é responsável pela implementação do modo de operação XTS, com suas funções de leitura e escrita de um bloco de dados.

É instanciada a partir da definição do algoritmo de encriptação e de duas senhas (uma do algoritmo e outra do XTS - para maiores detalhes do funcionamento do XTS veja a seção 2.2.10).

Também implementa uma função de encriptação e decriptação de múltiplos blocos, para facilitar a compatibilidade com o TrueCrypt, já que o tamanho básico do bloco XTS é menor que o do TrueCrypt (o bloco básico do XTS é de 128 bits, enquanto o TrueCrypt utiliza blocos de 512 bytes).

Gf2n

Essa classe é auxiliar à classe *Xts*, e implementa a função de multiplicação polinomial binária em *Galois Field* (campo finito) necessária para o XTS.

Além da função genérica de multiplicação, existe uma função específica que facilita a compatibilidade com o TrueCrypt, utilizando módulo 2^{128}.

Como demonstrado na seção 2.2.10, que explica o funcionamento do XTS, a multiplicação binária é utilizada para fortalecer o embaralhamento dos dados dentro do bloco.

O ícone do aplicativo criado foi baseado no logotipo do TrueCrypt e é demonstrado na Figura 4.3 a seguir.

Figura 4.3 - Ícone mtCrypt

Fonte: elaborado pelos autores.

Nas subseções seguintes são apresentadas algumas considerações importantes sobre o desenvolvimento do projeto.

4.2.1 Dificuldades encontradas

Alguns contratempos encontrados ao longo do desenvolvimento do projeto são apresentados a seguir. Foram ocorrências que não estavam no planejamento inicial e geraram problemas imprevistos. Suas soluções precisaram ser avaliadas e inseridas no cronograma com prioridade para não atrapalharem o andamento do projeto.

1. Necessidade de implementação do XTS

No início do projeto, assumimos que seria possível achar todas as primitivas criptográficas já implementadas em alguma biblioteca, e bastaria importá-la para utilizar suas funções. Na maior parte dos casos isso se mostrou verdadeiro, pois o gnu-crypto implementa em Java a maior parte dessas funções (veja a seção 4.1.3).

O que não estava presente no gnu-crypto é a implementação do modo de operação XTS. Não localizamos nenhuma biblioteca em Java que implementava essa função e, portanto, foi necessário desenvolver uma solução própria. O que localizamos com nossas buscas foram algumas implementações em outras linguagens, que por causa disso não seriam possíveis de serem reutilizadas no nosso projeto.

Tomando como base essas implementações em outras linguagens e a definição conceitual do modo de funcionamento, desenvolvemos nossa solução de XTS e validamos seu funcionamento ao testar os resultados com essas outras implementações e também com o TrueCrypt original.

2. Processamento lento

Em determinado estágio do desenvolvimento do aplicativo nos deparamos com um problema de desempenho relacionado à encriptação e decriptação de um volume de dados. Essas funções passaram a demorar vários minutos para serem concluídas, o que reduzia a qualidade da usabilidade do aplicativo para o usuário.

Verificamos que havia um problema de gerenciamento de memória por parte da nossa implementação do XTS. As classes que utilizávamos consumiam muita memória e sobrecarregavam o *garbage collector* da JVM, causando assim o baixo desempenho do aplicativo.

Refatoramos a implementação do XTS com atenção extra à questão do desempenho e da utilização de memória. Conseguimos reescrever a solução de forma mais otimizada e adicionamos um processo em *background* que adianta algumas tarefas necessárias.

3. Utilização do FUSE

A utilização do FUSE neste projeto serve basicamente para deixar transparente ao usuário as funções de leitura e escrita de arquivos, utilizados na importação e exportação deles.

Sem o FUSE, seria necessário encriptar ou decriptar um arquivo, construí-lo na memória RAM e, só depois de completo, escrevê-lo no dispositivo de armazenamento. Com o FUSE, é possível fazer todo esse processo dentro de uma única chamada de escrita ou leitura do próprio sistema de arquivos, que encapsularia a encriptação ou decriptação do arquivo junto com a escrita ou leitura no dispositivo de armazenamento, dessa forma economizando etapas do processo e reduzindo a possibilidade de brechas de segurança ao retirar do fluxo de processos a memória RAM.

4.2.2 Adaptações em relação à especificação

Devido às dificuldades encontradas, foi necessário fazer alguma adaptações no projeto original especificado na seção 3. São pendências de desenvolvimento que, se houvesse mais tempo ou mais pessoas trabalhando no projeto, talvez não existissem.

1. Interface de utilização

O desenho da interface de utilização (telas do aplicativo) não foi seguido à risca como especificado, e foi implementado para conter apenas o suficiente de informações e funcionalidades das telas previstas.

A motivação para essa adaptação foi o foco prioritário nas funções do aplicativo, em detrimento da percepção de beleza visual das telas. Desta forma, garantimos a funcionalidade do aplicativo e deixamos como pendências os acertos visuais da interface.

Por exemplo, quanto à tela de navegação para seleção de volume, utilizamos o próprio navegador de arquivos do Android, ao invés de implementar um navegador próprio. Dessa forma, economizamos tempo de projeto.

2. Função de abrir um arquivo dentro do volume

Não será possível implementar a função de abrir um arquivo dentro do volume em tempo hábil. Identificamos vários problemas que precisariam ser solucionados, tanto em termos de implicações sobre a segurança dos dados envolvidos, quanto em termos de método de implementação.

Quanto à segurança, deveríamos garantir que o arquivo a ser aberto não deixaria vestígios de dados que poderiam ser utilizados para quebrar a encriptação do volume. Quanto à implementação, deveríamos criar uma interface de transferência de dados entre nosso aplicativo e o aplicativo responsável por abrir o arquivo selecionado (por exemplo, um arquivo ".txt" e o editor de texto do usuário) que mantivesse a integridade dos dados do arquivo e ainda assim mantivesse a segurança do volume.

Como o aplicativo já contará com as funções de importar e exportar um arquivo, entendemos que se o usuário quiser manipular um determinado arquivo basta exportá-lo para a memória do telefone, fazer as alterações desejadas e importá-lo depois. O resultado seria o mesmo.

4.3 Testes

A seguir, são apresentados os testes realizados para a validação das funções do aplicativo. Os testes foram desenvolvidos com o objetivo de verificar se alterações

no código não quebravam nenhuma funcionalidade já implementada previamente, e se os resultados apresentados pelas funções estão de acordo com os resultados esperados.

testGf2nMul()

Esse teste avalia a implementação da função de multiplicação polinomial binária em *Galois Field*.

Figura 4.4 - testGf2nMul()

Fonte: elaborado pelos autores.

testGf2powMul()

Esse teste avalia a implementação da função de multiplicação polinomial binária em *Galois Field*, com módulo 2^{128}. Essa função utiliza a função base testada acima, porém o módulo específico serve para facilitar sua integração com o TrueCrypt.

Figura 4.5 - testGf2powMul()

```
@Test
public void testGf2powMul() {
    assertTrue(Gf2n.gf2pow128mul(
            new BigInteger("b9623d587488039f1486b2d8d9283453", 16),
            new BigInteger("a06aea0265e84b8a", 16)).toString(16)
                .equals("fead2ebe0998a3da7968b8c2f6dfcbd2"));
    assertTrue(Gf2n.gf2pow128mul(
            new BigInteger("0696ce9a49b10a7c21f61cea2d114a22", 16),
            new BigInteger("8258e63daab974bc", 16)).toString(16)
                .equals("89a493638cea727c0bb06f5e9a0248c7"));
    assertTrue(Gf2n.gf2pow128mul(
            new BigInteger("ecf10f64ceff084cd9d9d1349c5d1918", 16),
            new BigInteger("f48a39058af0cf2c", 16)).toString(16)
                .equals("80490c2d2560fe266a5631670c6729c1"));
    assertTrue(Gf2n.gf2pow128mul(
            new BigInteger("9c65a83501fae4d5672e54a3e0612727", 16),
            new BigInteger("9d8bc634f82dfc78", 16)).toString(16)
                .equals("d0c221b4819fdd94e7ac8b0edc0ab2cb"));
}
```

Fonte: elaborado pelos autores.

textXtsDecrypt()

Esse teste simula a decriptação de um cabeçalho de um volume TrueCrypt e verifica se o cabeçalho final decriptado está igual ao cabeçalho esperado.

Figura 4.6 - textXtsDecrypt()

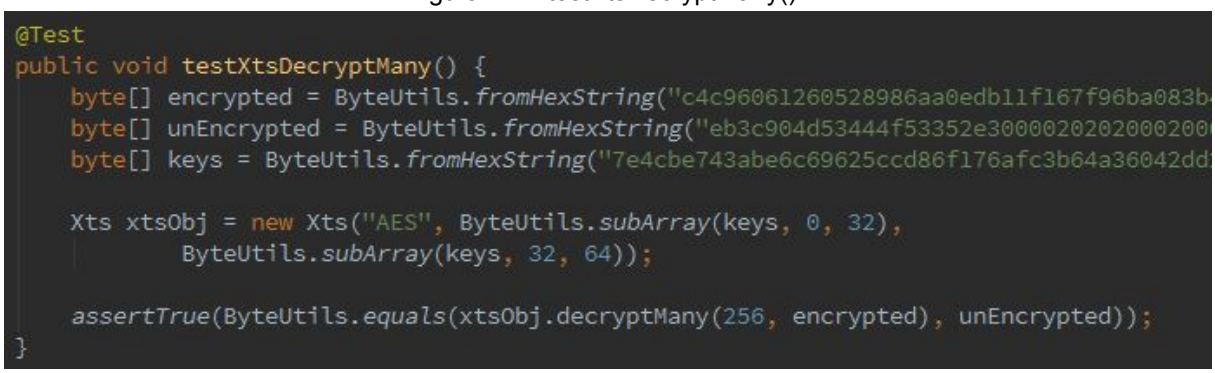
```
@Test
public void textXtsDecrypt() {
    byte[] encryptedTcHeader = ByteUtils.fromHexString("7f38134137fa1fd2c9a5b425851761
    byte[] unEncryptedTcHeader = ByteUtils.fromHexString("5452554500050700 86da92a50000

    String pass = "mtcrypt";

    byte[] salt = ByteUtils.fromHexString("e21d652d449e66beae81d4414852ef062de2b5e9453

    PKCS5S2ParametersGenerator generator = new PKCS5S2ParametersGenerator(
            new RIPEMD160Digest());
    generator.init(PBEParametersGenerator.PKCS5PasswordToUTF8Bytes(
            pass.toCharArray()), salt, 2000);
    KeyParameter key = (KeyParameter) generator.generateDerivedMacParameters(1536);

    Xts xtsObj = new Xts("AES", ByteUtils.subArray(key.getKey(), 0, 32),
            ByteUtils.subArray(key.getKey(), 32, 64));

    assertTrue(ByteUtils.equals(xtsObj.decryptMany(0, encryptedTcHeader),
            unEncryptedTcHeader));
}
```

Fonte: elaborado pelos autores.

testXtsDecryptMany()

Esse teste simula a decriptação de um bloco de dados e verifica se o resultado da decriptação está igual ao bloco de dados esperado.

Figura 4.7 - testXtsDecryptMany()



```
@Test
public void testXtsDecryptMany() {
    byte[] encrypted = ByteUtils.fromHexString("c4c960612605289 86aa0edb11f167f96ba083b
    byte[] unEncrypted = ByteUtils.fromHexString("eb3c904d53444f53352e3000020202000200
    byte[] keys = ByteUtils.fromHexString("7e4cbe743abe6c69625ccd86f176afc3b64a36042dd

    Xts xtsObj = new Xts("AES", ByteUtils.subArray(keys, 0, 32),
            ByteUtils.subArray(keys, 32, 64));

    assertTrue(ByteUtils.equals(xtsObj.decryptMany(256, encrypted), unEncrypted));
}
```

Fonte: elaborado pelos autores.

textXtsEncrypt()

Esse teste simula a encriptação de um cabeçalho de um volume TrueCrypt e verifica se o cabeçalho final encriptado está igual ao cabeçalho esperado.

Figura 4.8 - textXtsEncrypt()

```
@Test
public void textXtsEncrypt() {
    byte[] encryptedTcHeader = ByteUtils.fromHexString("7f38134137fa1fd2c9a5b425851761
    byte[] unEncryptedTcHeader = ByteUtils.fromHexString("5452554500050700 86da92a50000

    String pass = "mtcrypt";

    byte[] salt = ByteUtils.fromHexString("e21d652d449e66beae81d4414852ef062de2b5e9453

    PKCS5S2ParametersGenerator generator = new PKCS5S2ParametersGenerator(
            new RIPEMD160Digest());
    generator.init(PBEParametersGenerator.PKCS5PasswordToUTF8Bytes(
            pass.toCharArray()), salt, 2000);
    KeyParameter key = (KeyParameter) generator.generateDerivedMacParameters(1536);

    Xts xtsObj = new Xts("AES", ByteUtils.subArray(key.getKey(), 0, 32),
            ByteUtils.subArray(key.getKey(), 32, 64));

    assertTrue(ByteUtils.equals(xtsObj.encryptMany(0, unEncryptedTcHeader),
            encryptedTcHeader));
}
```

Fonte: elaborado pelos autores.

testXtsEcryptMany()

Esse teste simula a encriptação de um bloco de dados e verifica se o resultado da encriptação está igual ao bloco de dados esperado.

Figura 4.9 - testXtsEcryptMany()

```
@Test
public void testXtsEcryptMany() {
    byte[] encrypted = ByteUtils.fromHexString("c4c96061260528986aa0edb11f167f96ba08
    byte[] unEncrypted = ByteUtils.fromHexString("eb3c904d53444f53352e30000202020002

    byte[] keys = ByteUtils.fromHexString("7e4cbe743abe6c69625ccd86f176afc3b64a36042

    Xts xtsObj = new Xts("AES", ByteUtils.subArray(keys, 0, 32),
            ByteUtils.subArray(keys, 32, 64));

    assertTrue(ByteUtils.equals(xtsObj.encryptMany(256, unEncrypted), encrypted));
}
```

Fonte: elaborado pelos autores.

testFat()

Esse teste verifica se um volume TrueCrypt foi decriptado e montado corretamente como um sistema de arquivos FAT32.

Figura 4.10 - testFat()

```
@Test
public void testFat() {
    String pass = "mtcrypt";
    File volume = getFileFromPath(this, "res/testTc_ripmed160_aes256.tc");
    try {
        tcVolume v = new tcVolume(pass, volume);
        v.decryptHeader();
        if (v.isDecrypted) {
            tcBlockDevice bDev = new tcBlockDevice(v.getVolume(), true,
                    v.getAlgorithm(), v.getKeys(), v.volStart, v.volEnd);
            FatFileSystem fs = FatFileSystem.read(bDev, true);
            assertTrue(fs.getRoot().iterator().next().getName().equals("externalVolDir"));
        } else {
            // Fail test
            assertTrue(false);
        }
    } catch (Exception e) {
        e.printStackTrace();
        assertTrue(false);
    }
}
```

Fonte: elaborado pelos autores.

testFatHiddenVol()

Esse teste verifica se um volume TrueCrypt oculto foi decriptado e montado corretamente como um sistema de arquivos FAT32.

Figura 4.11 - testFatHiddenVol()

```
@Test
public void testFatHiddenVol() {
    String pass = "mtcrypthidden";
    File volume = getFileFromPath(this, "res/testTc_ripmed160_aes256.tc");
    try {
        tcVolume v = new tcVolume(pass, volume);
        v.decryptHeader();
        if (v.isDecrypted) {
            tcBlockDevice bDev = new tcBlockDevice(v.getVolume(), true,
                    v.getAlgorithm(), v.getKeys(), v.volStart, v.volEnd);
            FatFileSystem fs = FatFileSystem.read(bDev, true);
            assertTrue(fs.getRoot().iterator().next().getName().equals("hiddenVolDir"));
        } else {
            // Fail test
            assertTrue(false);
        }
    } catch (Exception e) {
        e.printStackTrace();
        assertTrue(false);
    }
}
```

Fonte: elaborado pelos autores.

testBlockDeviceWrite()

Esse teste verifica a função de escrita e leitura da classe *tcBlockDevice*, verificando se os dados escritos são posteriormente lidos corretamente.

Figura 4.12 - testBlockDeviceWrite()

```
@Test
public void testBlockDeviceWrite() {
    String pass = "mtcrypt";
    File volume = getFileFromPath(this, "res/testBDevWrite");

    byte[] byteArray = ByteUtils.fromHexString("5452554500050700 86da92a500000000000000000000
    byte[] salt = ByteUtils.fromHexString("e21d652d449e66beae81d4414852ef062de2b5e9453c352

    PKCS5S2ParametersGenerator generator = new PKCS5S2ParametersGenerator(
            new RIPEMD160Digest());
    generator.init(PBEParametersGenerator.PKCS5PasswordToUTF8Bytes(
            pass.toCharArray()), salt, 2000);
    KeyParameter key = (KeyParameter) generator.generateDerivedMacParameters(1536);

    try {
        tcBlockDevice bDev = new tcBlockDevice(volume, false, "AES", key.getKey(),
                0, volume.getTotalSpace() - 1);
        ByteBuffer buffer;
        buffer = ByteBuffer.allocate(byteArray.length);
        buffer.put(byteArray);
        bDev.write(0, buffer);
        buffer = ByteBuffer.allocate(byteArray.length);
        bDev.read(0, buffer);

        assertTrue(ByteUtils.equals(buffer.array(), byteArray));

        buffer = ByteBuffer.allocate(1);
        buffer.put(ByteUtils.fromHexString("55"));
        bDev.write(1, buffer);
        buffer = ByteBuffer.allocate(byteArray.length);
        bDev.read(0, buffer);

        assertTrue(ByteUtils.equals(buffer.array(), ByteUtils.fromHexString("5455554500050

        bDev.close();
    } catch (Exception e) {
        e.printStackTrace();
        assertTrue(false);
    }
}
```

Fonte: elaborado pelos autores.

5 CONCLUSÕES

Este projeto apresenta conceitos suficientes para o entendimento do TrueCrypt, tanto em termos de forma de funcionamento quanto em métodos de compatibilidade. As descrições conceituais sucintas da seção 2 preparam o entendimento das estratégias e métodos adotados na especificação do projeto planejado na seção 3. Após definição das tecnologias utilizadas, com base no alinhamento em relação ao objetivo do projeto e nas facilidades que elas proporcionariam ao longo de seu desenvolvimento, foi realizado de fato a implementação da solução.

Ao longo do período de implementação surgiram problemas inesperados e instigantes, que precisaram ser solucionados sem comprometer o andamento do projeto. Algumas decisões que foram tomadas desviaram o aplicativo final desenvolvido do projeto original, mas o resultado final foi satisfatório e cumpriu os objetivos essenciais do trabalho.

É importante ressaltar que ainda existem oportunidades de melhorias sobre o trabalho final, como apresentadas na seção 5.1 a seguir. Numa realidade em que soluções de segurança cada vez mais ganham destaque e importância na sociedade, as ferramentas de criptografia devem não só garantir a segurança dos dados do usuário, como também oferecer métodos fáceis e agradáveis de uso, que se integrem à sua rotina.

Em resumo, este trabalho cumpre seu objetivo proposto mas não identifica esse estado como último no seu ciclo de vida. As oportunidades de melhorias apresentam desafios a serem superados e, devido à crescente utilização de ferramentas de segurança pelos usuários de dispositivos móveis, definitivamente serão de grande valia num futuro próximo.

5.1 Trabalhos futuros

O mtCrypt, através de sua compatibilidade com o TrueCrypt, cumpre seu objetivo essencial de prover criptografia negável a qualquer usuário de dispositivo móvel com sistema operacional Android.

Entretanto, algumas funções que foram descartadas no planejamento ou abandonadas ao longo do projeto poderiam ser inseridas num trabalho mais elaborado. Por exemplo, a função de abrir e manipular um arquivo dentro do próprio ambiente encriptado do volume seria de grande valia para os usuários, pois embora o processo de exportar e importar arquivos acabe com o mesmo resultado, é um processo trabalhoso e consome tempo. Também poderia ser reavaliada a necessidade da permissão de superusuário ("*root*") para o funcionamento total do aplicativo, já que a grande maioria dos usuários de dispositivos móveis nem saibam que isso existe.

Outra consideração que pode ser feita é o desenvolvimento do projeto para outras plataformas e sistemas operacionais. Por exemplo, o sistema operacional iOS representa uma parcela considerável do mercado de dispositivos móveis e uma solução como o mtCrypt também seria útil para seus usuários.

Por fim, mas não menos importante, a interface de utilização do aplicativo poderia ser repensado e redesenhado com foco na facilidade de uso e na beleza visual dos elementos na tela.

REFERÊNCIAS

[1] TRUECRYPT. Disponível em: <http://truecrypt.sourceforge.net/>. Acessado em 15/08/2015.

[2] TRUECRYPT USER'S GUIDE. Disponível em: <https://www.grc.com/misc/truecrypt/TrueCrypt%20User%20Guide.pdf>. Acessado em 15/08/2015.

[3] TRUECRYPT UN-OFFICIAL REFERENCE SITE. Disponível em: <http://andryou.com/truecrypt/index.php>. Acessado em 15/08/2015.

[4] STALLINGS, W. Cryptography and Network Security - Principles and Practice. 6th Ed. Pearson, 2014. ISBN 10: 0-13-335469-5.

[5] PAAR, C.; PELZL, J. Understanding Cryptography - A Textbook for Students and Practitioners. Springer, 2010. ISBN: 978-3-642-04100-6.

[6] KLEIN, P. A Cryptography Primer. Cambridge University Press, 2014. ISBN: 978-1-107-01788-7.

[7] KATZ, J.; LINDELL, Y. Introduction to Modern Cryptography. 2nd Ed. CRC Press, 2015. ISBN: 978-1-4665-7027-6.

[8] BROZ, M.; MATYAS, V. The TrueCrypt On-Disk Format - An Independent View. IEEE Security and Privacy. Volume 12, Edição 3, maio/junho 2014. Artigo número 6824535, p. 74-77.

[9] MIAO, Q-X. Research and Analysis on Encryption Principle of TrueCrypt Software System. 2nd International Conference on Information Science and Engineering, ICISE 2010. Artigo número 5691392, p. 1409-1412.

[10] GUJRATI, S.; VASSERMAN, E. The Usability of TrueCrypt, or How I Learned to Stop Whining and Fix an Interface. 3rd ACM Conference on Data and Application Security and Privacy, CODASPY 2013. Código 96008, p. 83-93.

[11] RIJMEN, V.; DAEMEN, J. AES Proposal: Rijndael. National Institute of Standards and Technology. Disponível em <http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf>. Acessado em 05/09/2015.

[12] SERPENT. A Candidate Block Cipher for the Advanced Encryption Standard. Disponível em <http://www.cl.cam.ac.uk/~rja14/serpent.html>. Acessado em 05/09/2015.

[13] SCHNEIER ON SECURTY. Twofish. Disponível em <https://www.schneier.com/twofish.html>. Acessado em 05/09/2015.

[14] Report on the Development of the Advanced Encryption Standard (AES). Disponível em <http://csrc.nist.gov/archive/aes/round2/r2report.pdf>. Acessado em 05/09/2015.

[15] The hash function RIPEMD-160. Disponível em <http://homes.esat.kuleuven.be/~bosselae/ripemd160.html>. Acessado em 12/09/2015.

[16] SECURE HASHING. National Institute of Standards and Technology. Disponível em <http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html>. Acessado em 12/09/2015.

[17] Secure Hash Standard (SHS). National Institute of Standards and Technology. Disponível em <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>. Acessado em 12/09/2015.

[18] The WHIRLPOOL Hash Function. Disponível em <http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html>. Acessado em 12/09/2015.

[19] IEEE P1619™/D16. Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. Disponível em <http://libeccio.di.unisa.it/Crypto14/Lab/p1619.pdf>. Acessado em 13/09/2015.

[20] Microsoft Developer Network. FAT File System (Windows Embedded CE 6.0). Disponível em <https://msdn.microsoft.com/en-us/library/ee489982%28v=winembedded.60%29.aspx>. Acessado em 18/09/2015.

[21] PJRC. Understanding FAT32 Filesystems. Disponível em <https://www.pjrc.com/tech/8051/ide/fat32.html>. Acessado em 18/09/2015.

[22] blog.bjrn.se. TrueCrypt explained. Disponível em: <http://blog.bjrn.se/2008/01/truecrypt-explained.html>. Acessado em 21/08/2015.

[23] blog.bjrn.se. TrueCrypt explained (TrueCrypt 5 update). Disponível em: <http://blog.bjrn.se/2008/02/truecrypt-explained-truecrypt-5-update.html>. Acessado em 21/08/2015.

[24] Android. Disponível em <https://www.android.com/>. Acessado em 19/09/2015.

[25] Java. Disponível em <https://www.java.com/pt_BR/about/>. Acessado em 19/09/2015.

[26] Android Studio. Disponível em <https://developer.android.com/intl/pt-br/sdk/index.html>. Acessado em 19/09/2015.

[27] Android SDK. Disponível em <https://developer.android.com/intl/pt-br/sdk/installing/index.html>. Acessado em 19/09/2015.

[28] GNU Crypto Project. Disponível em <https://www.gnu.org/software/gnu-crypto/>. Acessado em 19/09/2015.

[29] FUSE - Filesystem in Userspace. Disponível em <http://fuse.sourceforge.net/>. Acessado em 19/09/2015.

[30] FAT32 Library. Disponível em <http://waldheinz.github.io/fat32-lib/>. Acessado em 19/09/2015.

[31] Git. Disponível em <https://git-scm.com/doc>. Acessado em 19/09/2015.

[32] GitHub. Disponível em <https://github.com/>. Acessado em 19/09/2015.

APÊNDICE A - Utilização do TrueCrypt no computador pessoal

Neste apêndice são descrito os passos para utilização do TrueCrypt no computador pessoal. Inicialmente, é explicado como criar um volume TrueCrypt. Logo em seguida, segue a explicação sobre como montar um volume já criado.

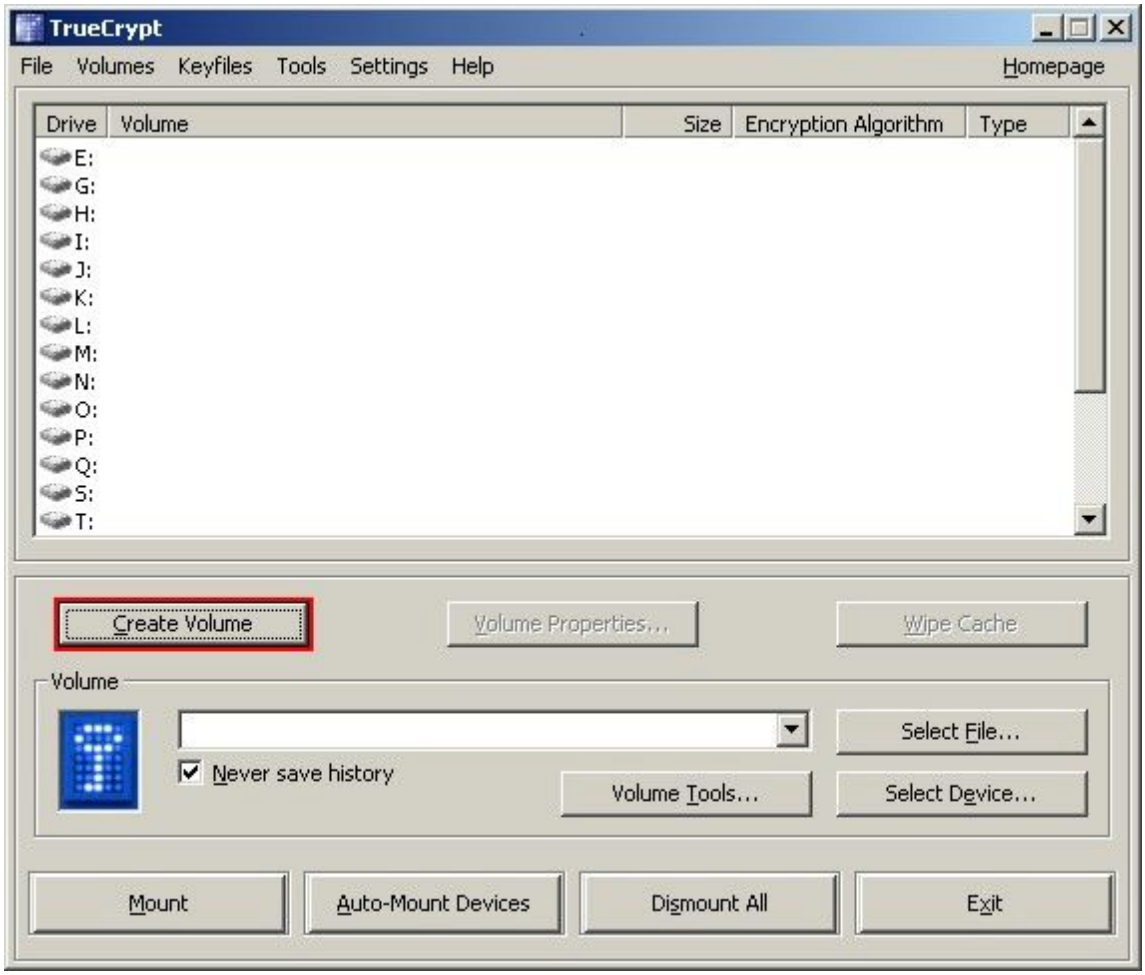
Passo 1:

Inicie o programa TrueCrypt. Se ele ainda não está instalado em seu computador, baixe o instalador e prossiga com a instalação.

Passo 2:

A tela principal do programa é mostrada na Figura A.1. Clique em "*Create Volume*" (destacado com um retângulo vermelho).

Figura A.1 - Passo 2

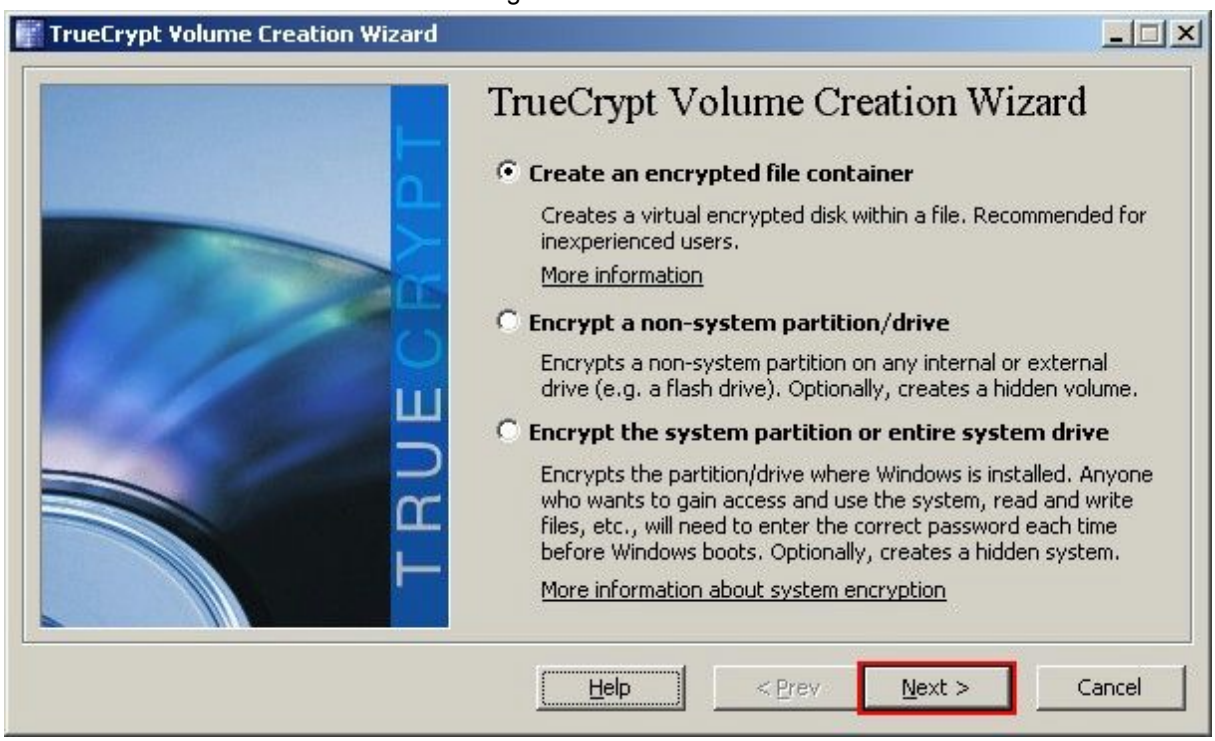


Fonte: [3].

Passo 3:

O utilitário de criação de volume aparecerá na tela, como mostrado na Figura A.2. Aqui, você escolhe o tipo de encriptação que quer. Neste guia criaremos um volume de arquivos armazenado num arquivo, então deixe selecionado a primeira opção e clique em "*Next*".

Figura A.2 - Passo 3



Fonte: [3].

Passo 4:

Neste passo, mostrado na Figura A.3, você escolhe se quer criar um volume padrão ou um volume oculto. Selecione o tipo padrão e clique em "*Next*".

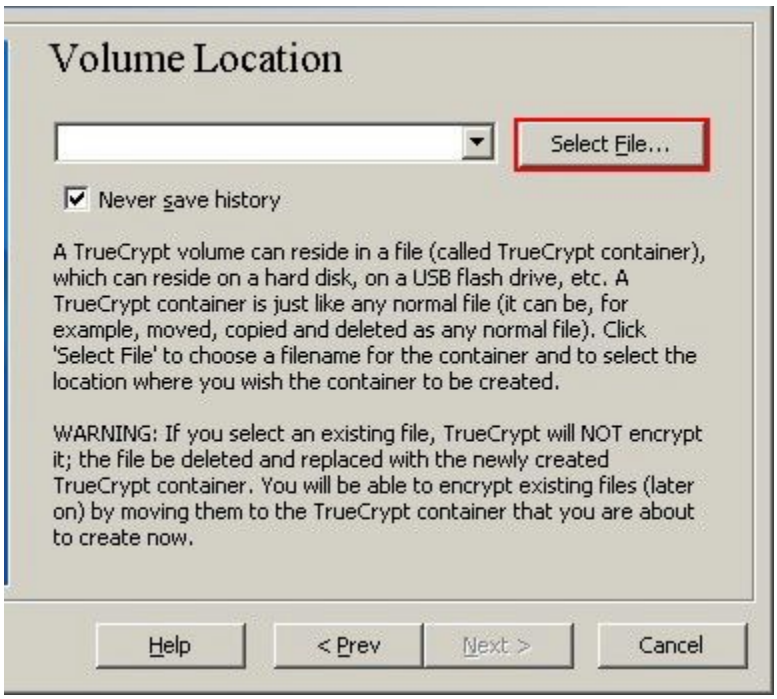
Figura A.3 - Passo 4

Fonte: [3].

Passo 5:

Neste passo, mostrado na Figura A.4, você especifica o arquivo que armazenará seu volume TrueCrypt. Note que ele será um arquivo como outro qualquer e pode ser copiado, renomeado, movido ou deletado. Clique em "*Select File...*".

Figura A.4 - Passo 5



Fonte: [3].

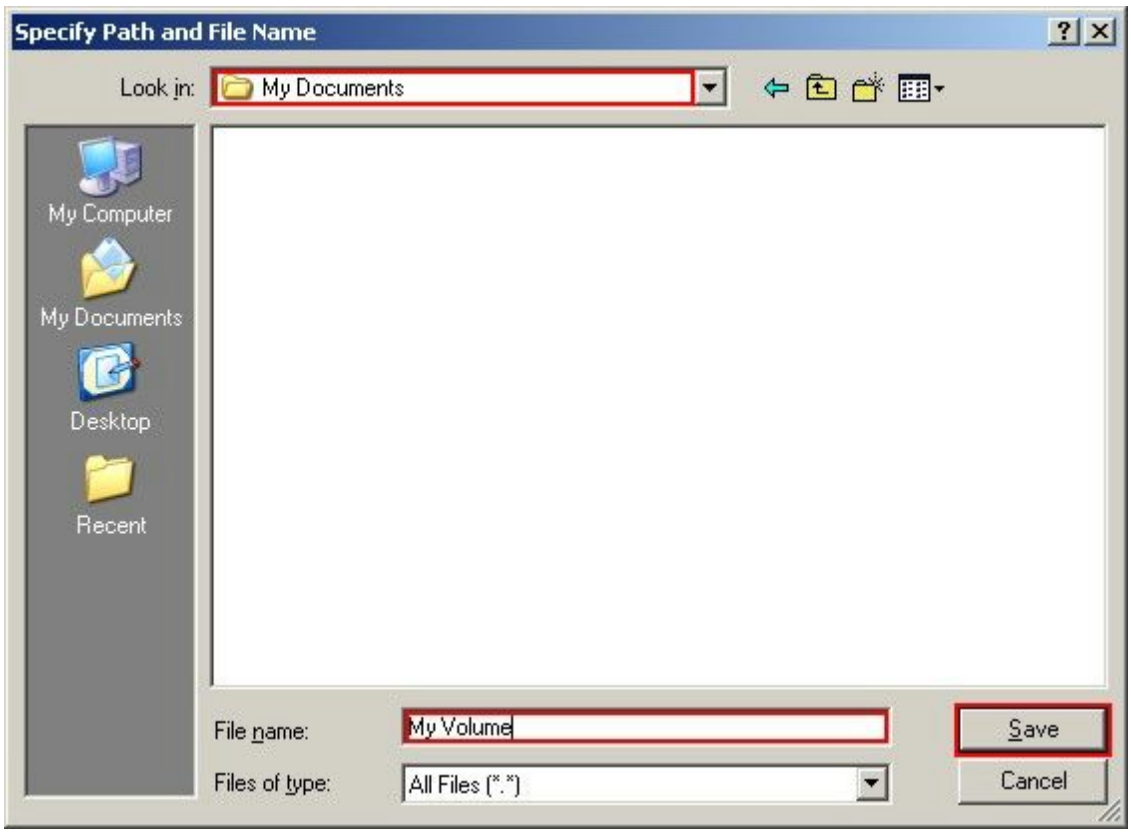
Passo 6:

A tela de seleção de arquivos aparecerá, como na Figura A.5. Neste guia, o arquivo TrueCrypt que será criado tem como caminho *D:\My Documents\My Volume*.

É importante ressaltar que, se um arquivo já existente for selecionado, ele será sobrescrito com os dados do volume TrueCrypt.

Selecione o local do arquivo desejado e dê um nome para o arquivo no campo "*File name*". Clique em "*Save*". Essa janela de seleção de arquivos deverá desaparecer.

Figura A.5 - Passo 6



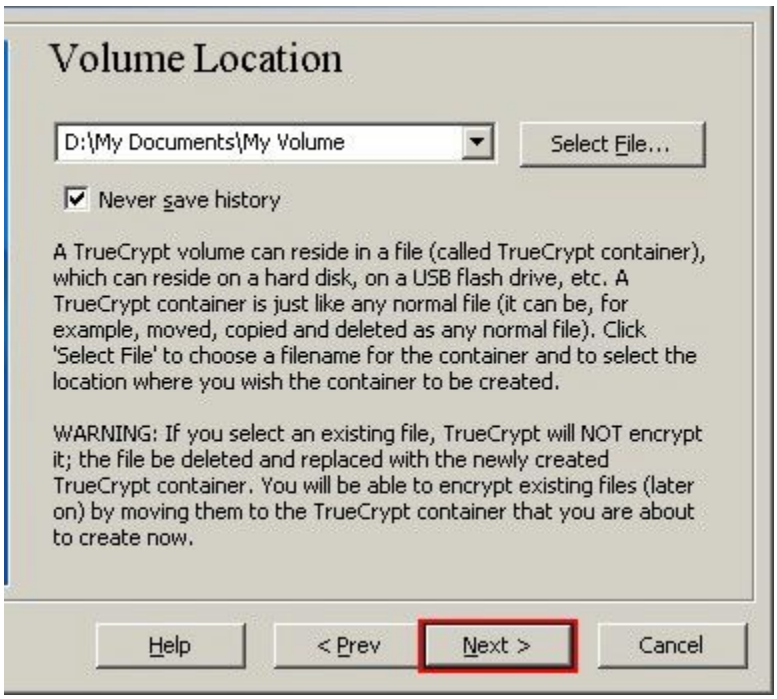
Fonte: [3].

Passo 7:

Na Figura A.6, vemos que o caminho do arquivo selecionado no passo anterior foi escrito no campo que estava em branco no passo 5.

Clique em "*Next*".

Figura A.6 - Passo 7

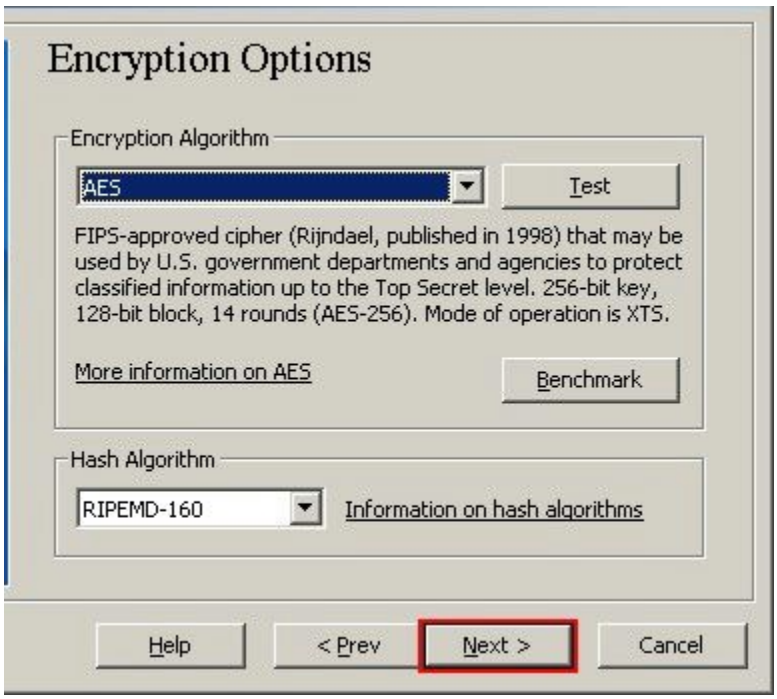


Fonte: [3].

Passo 8:

Nesta janela, mostrada na Figura A.7, você pode escolher o algoritmo de encriptação e a função de *hash* utilizados no volume. Após a escolha, clique em "*Next*".

Figura A.7 - Passo 8

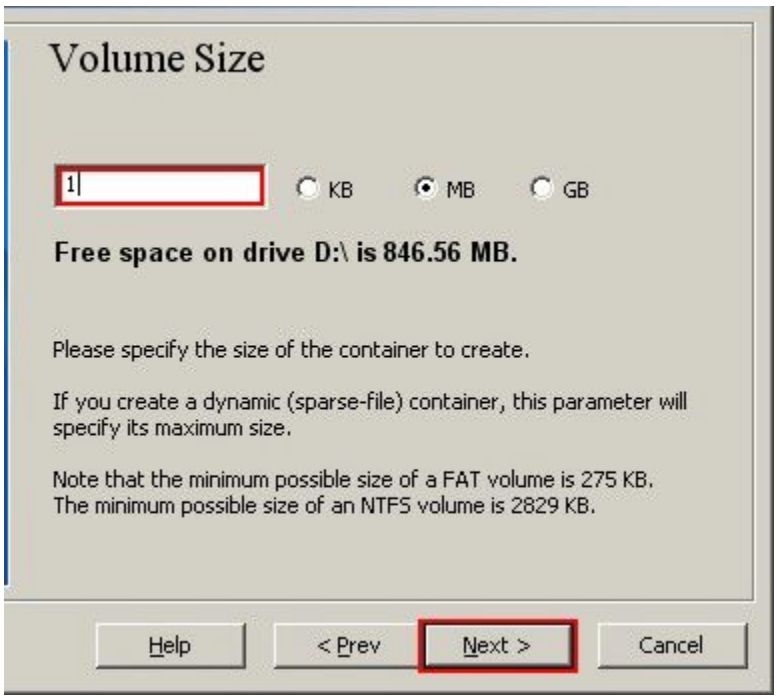


Fonte: [3].

Passo 9:

Nesta janela, mostrada na Figura A.8, você especifica o tamanho do seu volume TrueCrypt. Neste guia, utilizaremos o tamanho de 1 MB. Após a definição do tamanho do volume, clique em "*Next*".

Figura A.8 - Passo 9



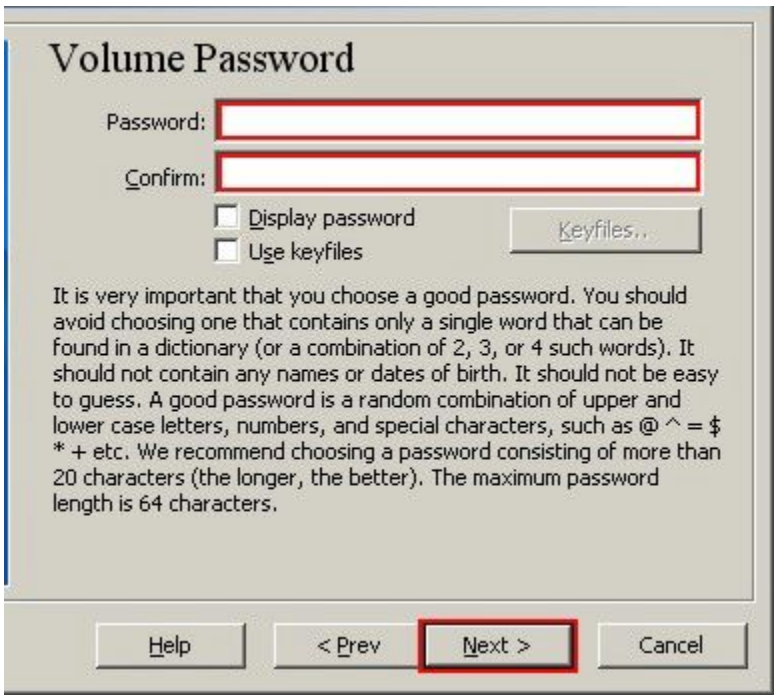
Fonte: [3].

Passo 10:

Nesta janela, mostrada na Figura A.9, você escolhe a senha do seu volume TrueCrypt. Recomenda-se uma senha grande, com a utilização de caracteres em maiúsculo, minúsculo, especiais e numéricos, todos em conjunto formando uma senha forte e difícil de ser quebrada.

Após a definição da senha e confirmação no campo abaixo, clique em "*Next*".

Figura A.9 - Passo 10



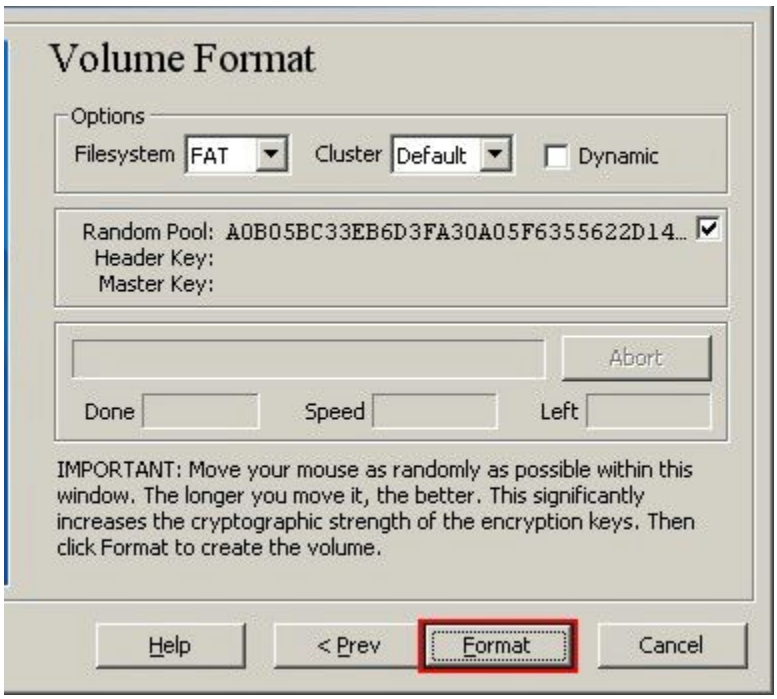
Fonte: [3].

Passo 11:

Nesta janela, mostrada na Figura A.10, é possível escolher o tipo do sistema de arquivos e o tamanho do *cluster* envolvido. Como poderá notar, ao movimentar o ponteiro do *mouse* o campo "*Random Pool*" se altera. Recomenda-se mover o ponteiro do *mouse* pela tela aleatoriamente, o que favorece a geração de senhas fortes de encriptação.

Após tudo isso, clique em "*Format*".

Figura A.10 - Passo 11.1



Fonte: [3].

Após isso, o volume TrueCrypt será criado e salvo no local indicado no passo 6.

Uma janela de confirmação, como a da Figura A.11, aparecerá no final do processo. Clique em "*OK*" para fechar a janela.

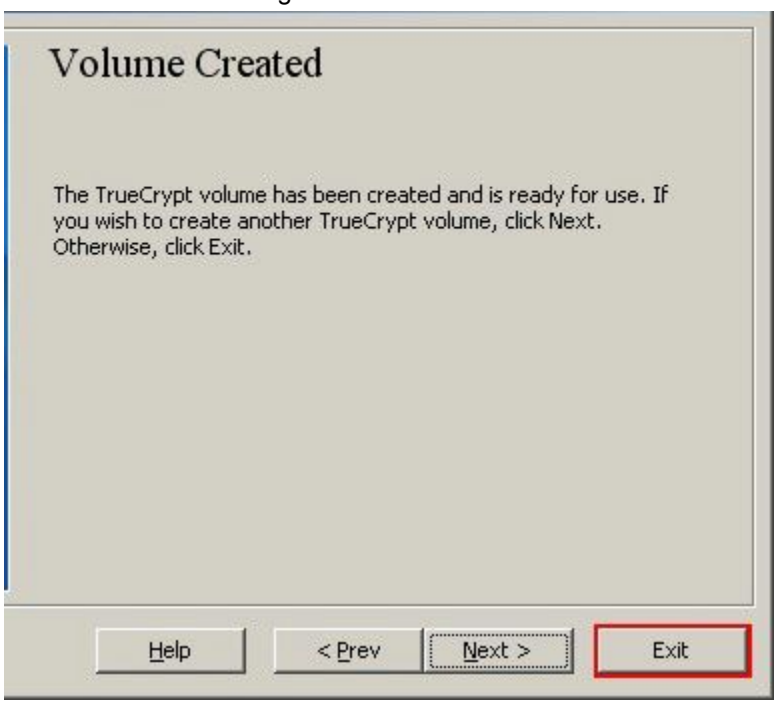
Figura A.11 - Passo 11.2

Fonte: [3].

Passo 12:

Nesta janela, mostrada na Figura A.12, aparece mais uma confirmação da criação do volume. Clique em "*Exit*" para fechar a janela e terminar o processo.

Figura A.12 - Passo 12



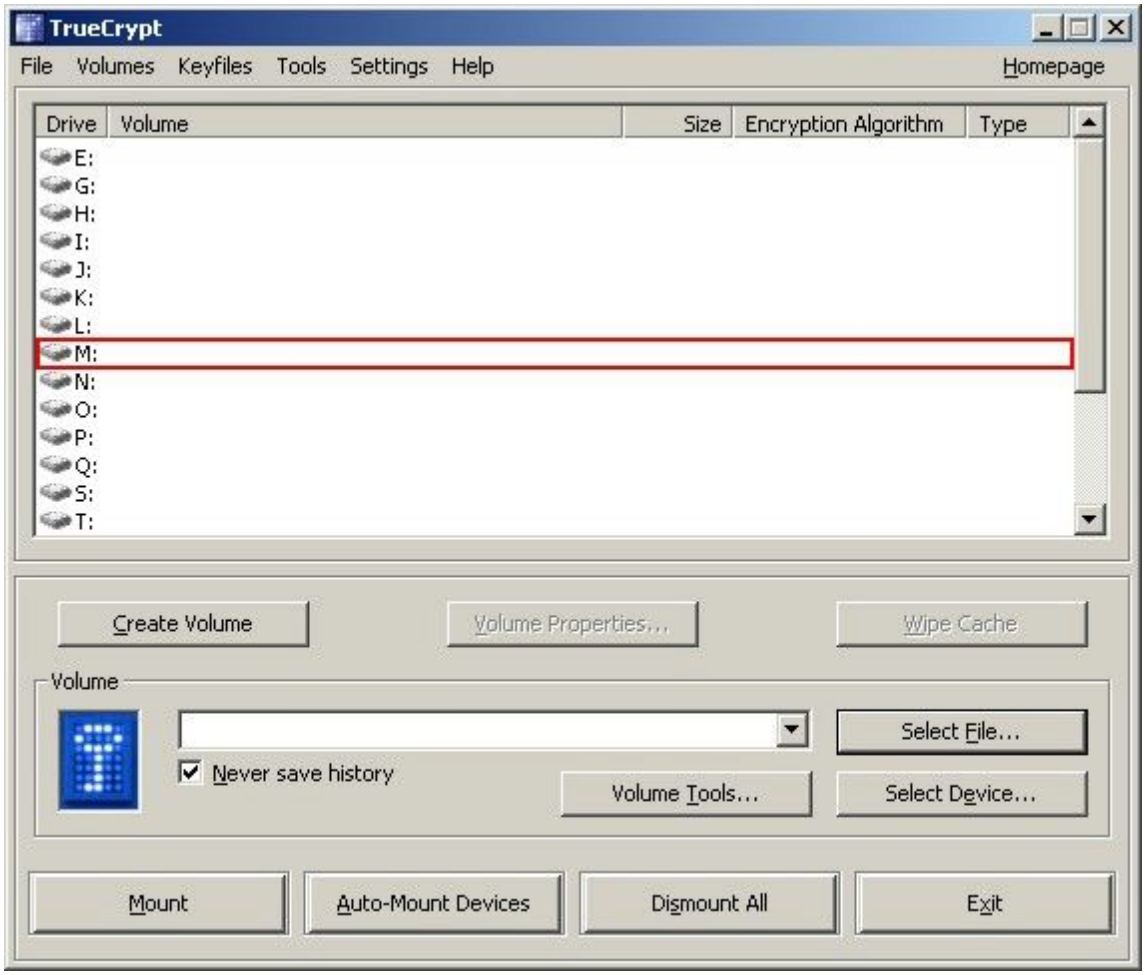
Fonte: [3].

Os passos seguintes descrevem como montar e utilizar um volume TrueCrypt. Tomaremos como base o volume que acabamos de criar neste guia.

Passo 13:

Esta janela, mostrada na Figura A.13, é novamente a tela inicial do programa. Desta vez, selecione uma letra de *drive* da lista. Neste guia selecionamos o *drive* de letra M, mas você pode escolher qualquer um disponível.

Figura A.13 - Passo 13

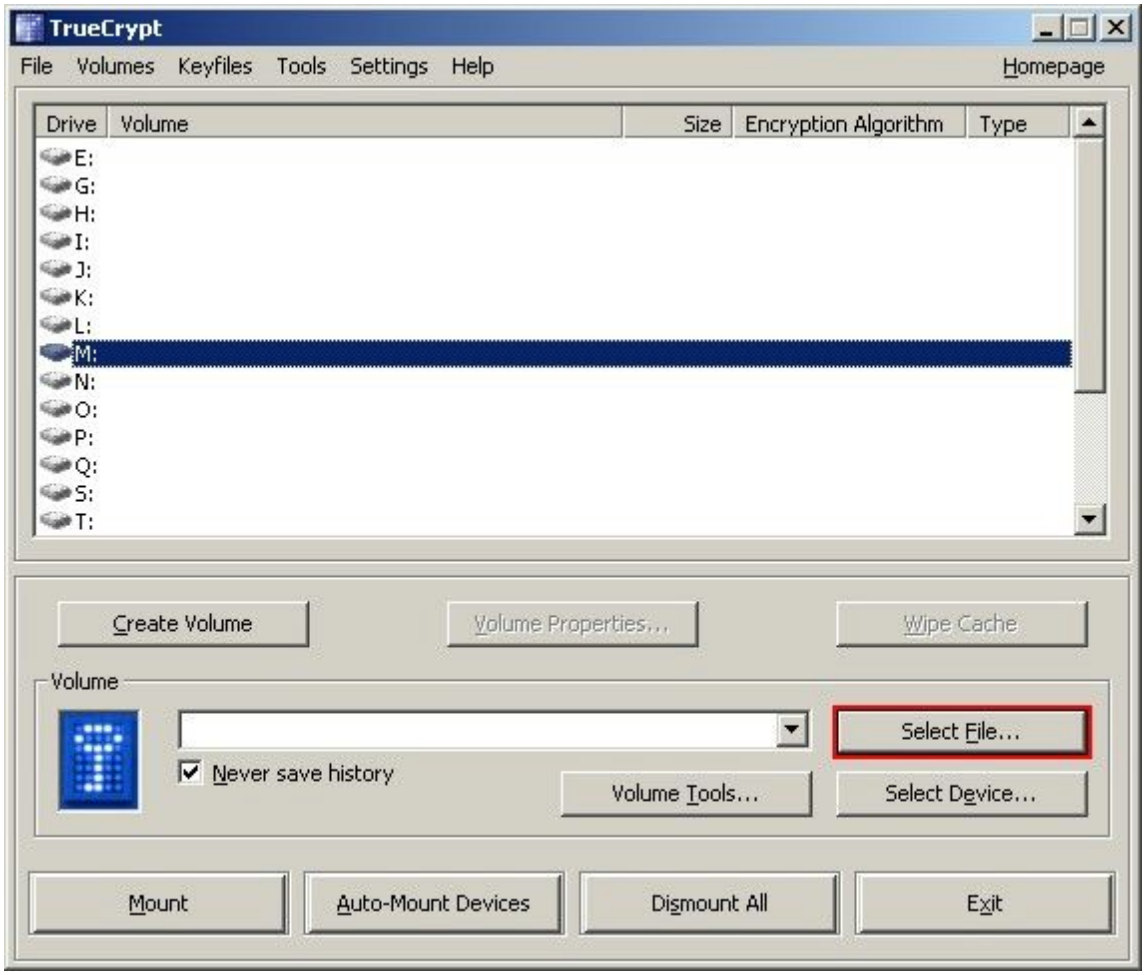


Fonte: [3].

Passo 14:

Após selecionado um *drive* da lista, clique em "*Select File...*", destacado com um retângulo vermelho na Figura A.14.

Figura A.14 - Passo 14

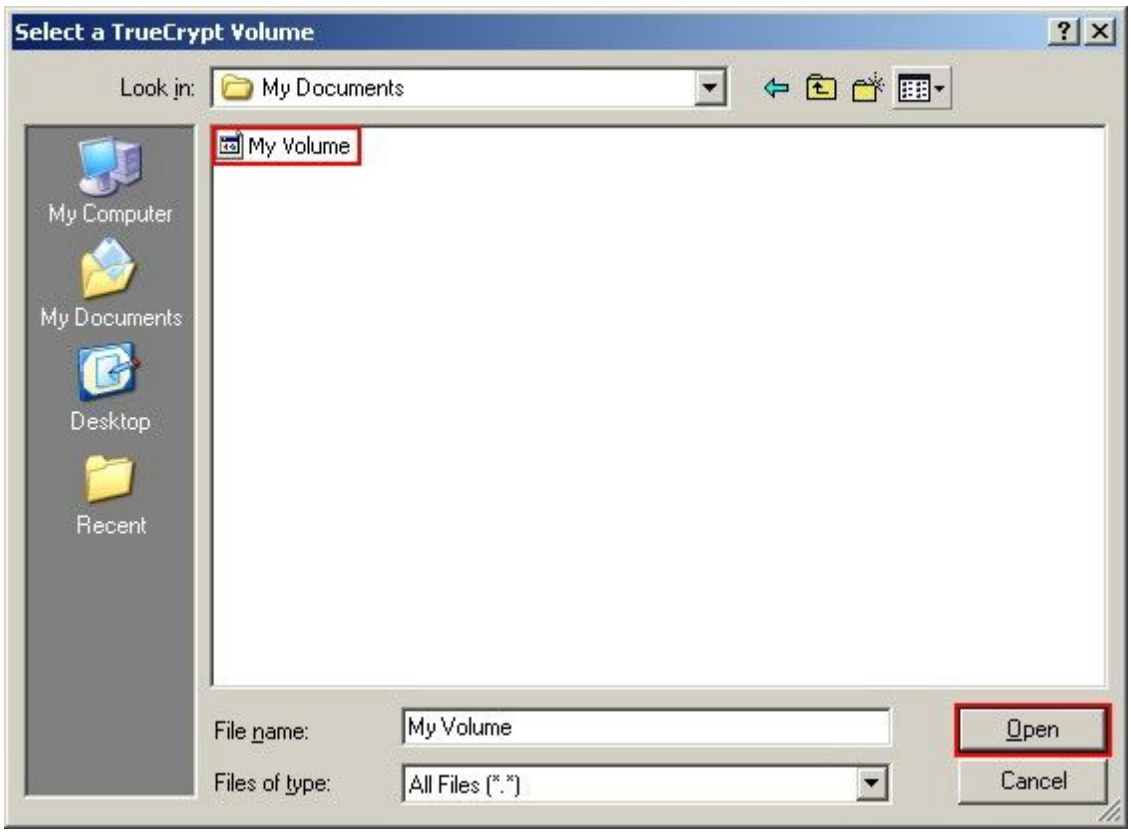


Fonte: [3].

Passo 15:

A janela de seleção de arquivos deve aparecer, como mostrado na Figura A.15. Selecione o arquivo de volume TrueCrypt e clique em "*Open*". A janela deve desaparecer e você retornará para a tela inicial do programa.

Figura A.15 - Passo 15

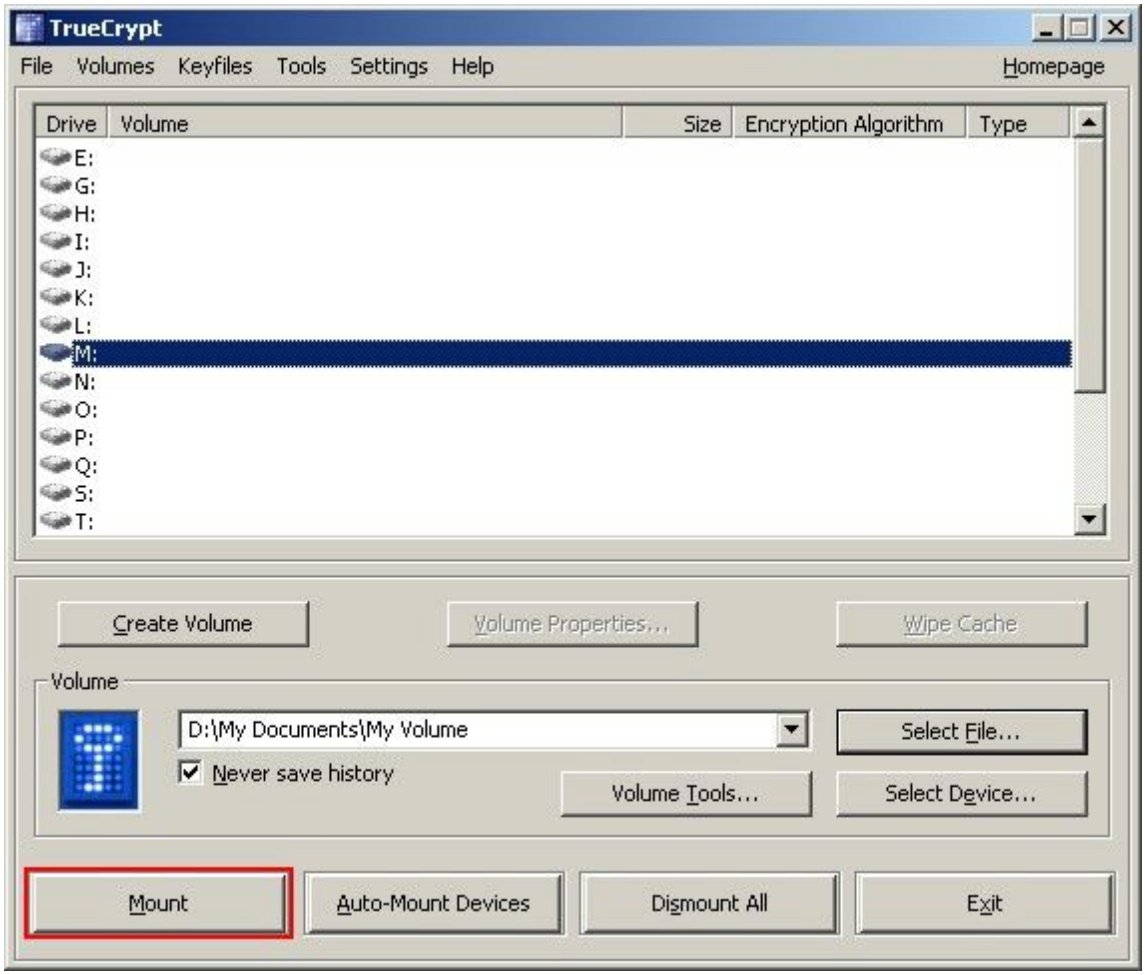


Fonte: [3].

Passo 16:

Como poderá notar, o caminho do arquivo selecionado preencheu o antigo campo em branco na seção inferior da janela, como mostrado na Figura A.16. Clique em "*Mount*".

Figura A.16 - Passo 16

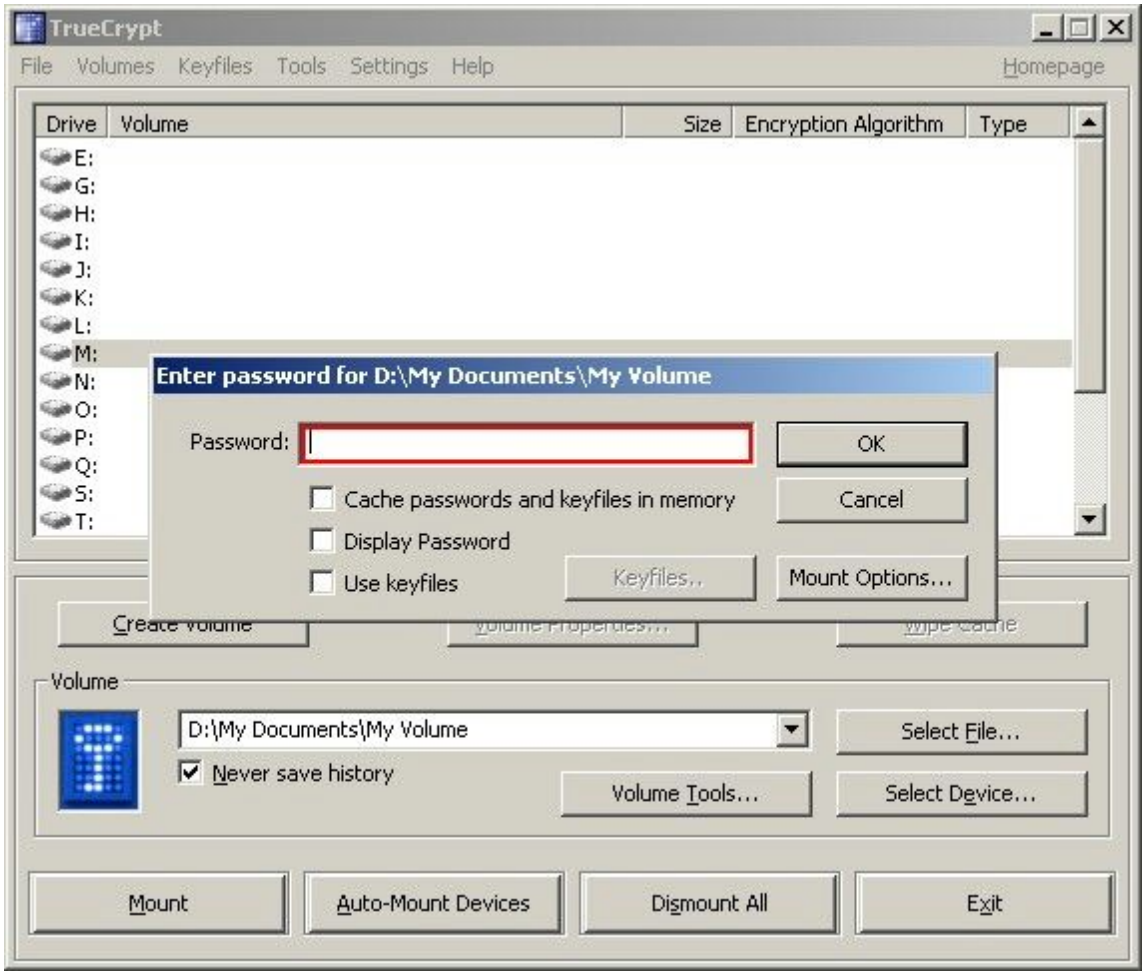


Fonte: [3].

Passo 17:

Uma janela aparecerá pedindo a senha do volume, como na Figura A.17.

Figura A.17 - Passo 17

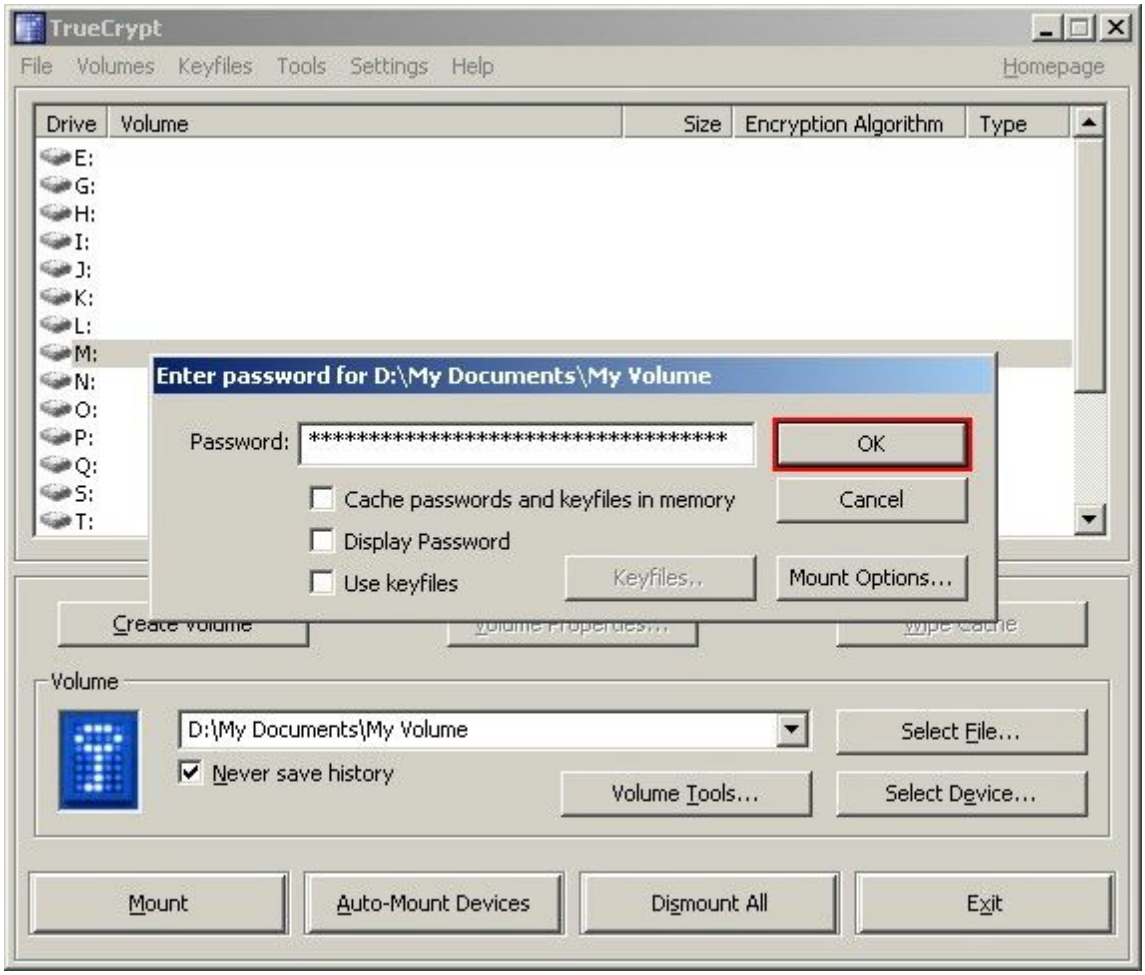


Fonte: [3].

Passo 18:

Entre com a senha correta e clique em “*OK*”, como na Figura A.18.

Figura A.18 - Passo 18



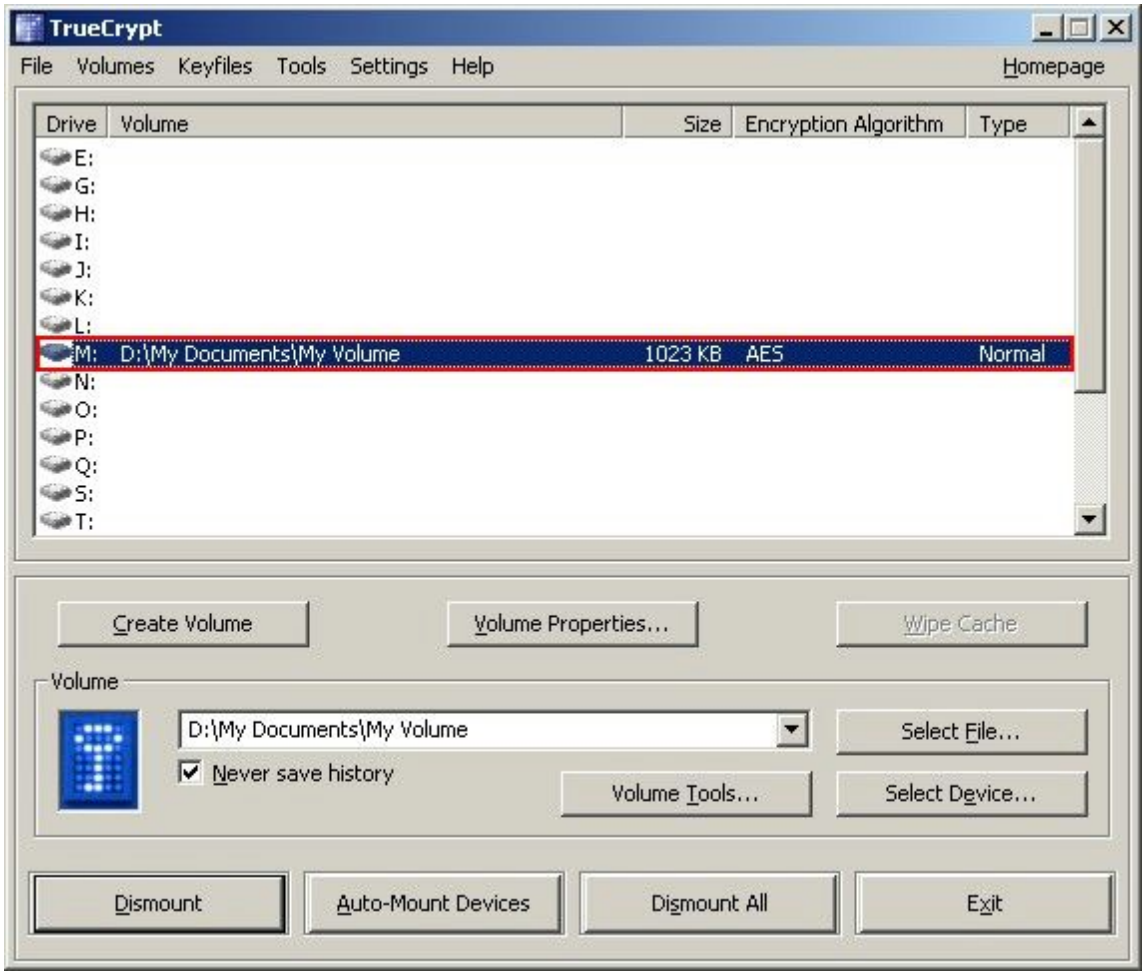
Fonte: [3].

Após isso, o TrueCrypt tentará decriptar e montar o volume no sistema de arquivos do computador. Se a senha fornecida não for correta, o programa notificará o erro e retornará para o passo anterior. Se for correta, o volume será decriptado e montado.

Passo final:

O volume foi decriptado e montado com sucesso no *drive* escolhido anteriormente. Isso é mostrado na Figura A.19.

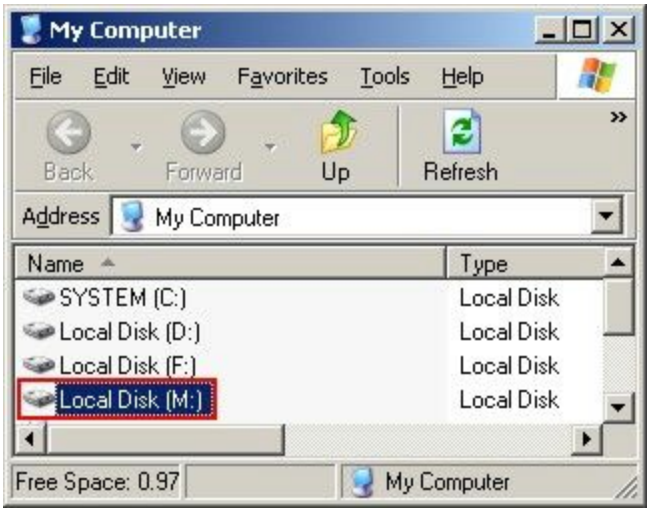
Figura A.19 - Passo final 1



Fonte: [3].

O volume é tratado como um disco virtual e tem como raiz a letra do *drive* selecionado na lista. Se você explorar seus arquivos do computador, poderá ver que o *drive* montado é identificado como um disco qualquer. Isso é mostrado na Figura A.20.

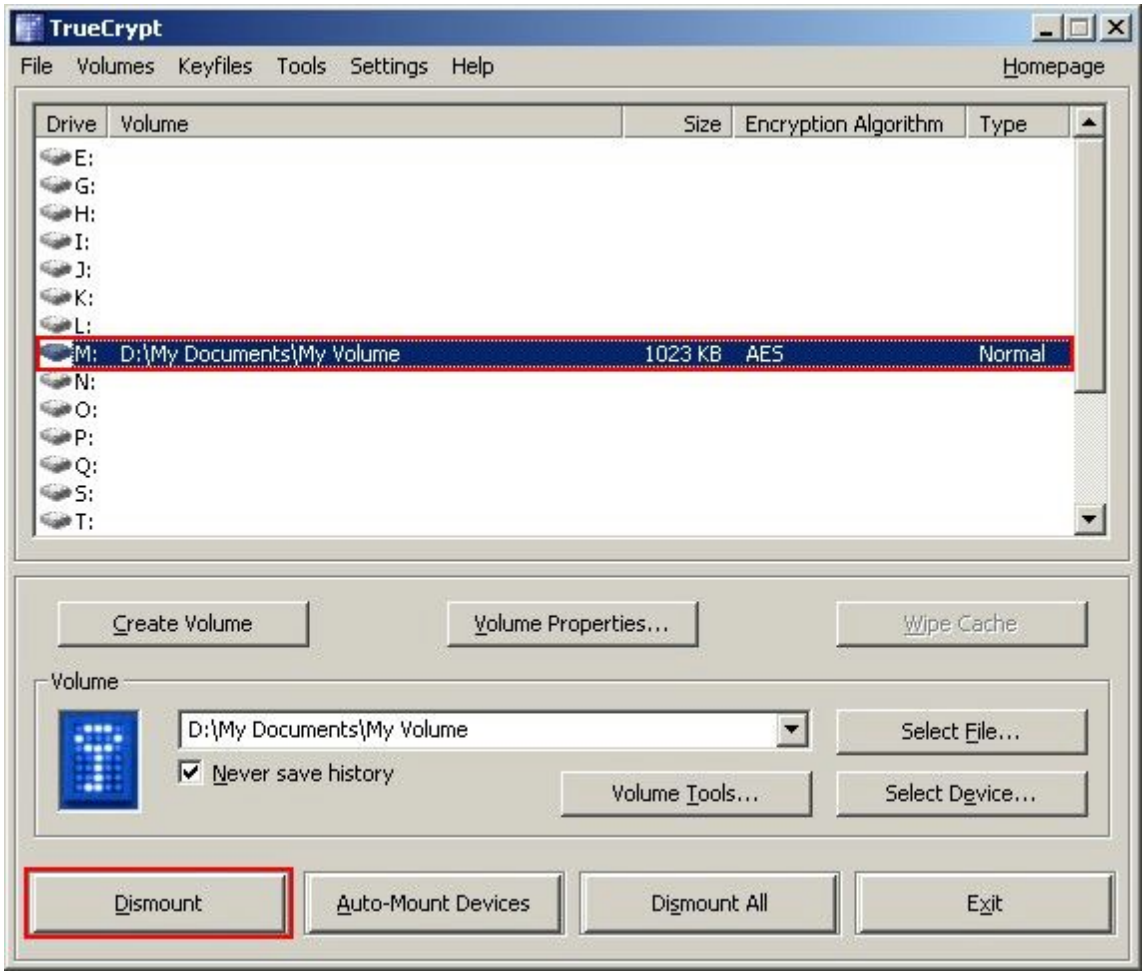
Figura A.20 - Passo final 2



Fonte: [3].

Por fim, após realizar as devidas alterações desejadas no seu volume de dados TrueCrypt, basta desmontá-lo para esconder a existência dos arquivos ali dentro. Para desmontar um volume, retorne à janela principal do TrueCrypt, selecione o *drive* montado e clique em "*Dismount*", destacado na Figura A.21.

Figura A.21 - Passo final 3



Fonte: [3].