Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Alessandro Wagner Palmeira
Caio Renato Bedulli do Carmo

Estudo de criptografia para HDDs e SSDs

São Paulo
Dezembro de 2017

Estudo de criptografia para HDDs e SSDs

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Marco Dimas Gubitoso

São Paulo
Dezembro de 2017

Resumo

Segurança de informação é um tema recorrente em nossa sociedade há séculos. Sendo assim, o presente trabalho visa o estudo de dois métodos que objetivam *Full Disk Encryption* (FDE): *cryptsetup* em conjunto com *Linux Unified Key Setup* (LUKS) e o utilitário SEDutil (interface para OPAL) e suas interações com o Sistema Operacional e a máquina.

Disponibilizamos diferenças no desempenho quando testados esses métodos com os dispositivos HDD e SSD de uso próprio, vulnerabilidades as quais estão expostas como ataque DMA, *Cold Boot* e *Evil Maid* e como reduzir tais exposições.

Quanto ao teste de desempenho, em nosso SSD, obtivemos uma redução de aproximadamente 8% na escrita e menor que 0.1% na leitura. Já em nosso HDD, obtivemos reduções variáveis entre 18,2% e 22,6% com suporte a AES e entre 17,28% e 23,73% sem suporte a AES na escrita. Para leitura, as reduções variaram entre 0% e 21% com suporte e 7% e 21% sem suporte.

No que tange a segurança, nossa revisão literária mostrou que ambos métodos são vulneráveis se não houver outra medida de prevenção usada em conjunto como BitVisor, Qubes OS, senha na BIOS para inicialização, suporte na CPU de IOMMU e TPM.

Palavras-chave: HDD, SSD, Criptografia, LUKS, SEDutil, Vulnerabilidade, Desempenho.

Abstract

Information security has been an issue in our society for centuries. Based on that, the present work aims the study of two methods focused on Full Disk Encryption (FDE): cryptsetup combined with Linux Unified Key Setup (LUKS) and the SEDutil application (OPAL interface) and their interactions with eh Operating System and the machine.

Performance differences have been shown when those methods were used on our personal HDD and SSD, vulnerabilities which those devices are exposed to like DMA, Cold Boot and Evil Maid and how to mitigate such attacks.

Regarding the performance test, on our SSD, there was a decrease of about 8% on writing rate and less than 0.1% in reading rate. As for our HDD, there were varying decreases between 18,2% and 22,6% when used native AES support and 17,28% and 23,73% without native AES support for writing task. On the reading task, the decreases varied between 0% and 21% when used AES native support and 7% e 21% without native AES support.

Our literature review has demonstrated that both methods are vulnerable if no additional preventing measure is used altogether such as BitVisor, Qubes OS, BIOS bootstrap password, IOMMU CPU support and TPM.

Keywords: HDD, SSD, Cryptography, LUKS, SEDutil, Vulnerability, Performance.

Sumário

Lista de Abreviaturas

ACPI	*Advanced Configuration and Power Interface*
AES	*Advanced Encryption Standard*
AMD	Sigla da empresa *Advanced Micro Devices*
AMD-V	*AMD Virtualization*
API	*Application Programming Interface*
ATA	*Advanced Technology Attachment*
BIOS	*Basic InputOutput System*
BSD	Sistemas operacionais distribuídos sob a *Berkeley Software Distribution*
CPU	Unidade Central de Processamento (*Central Processing Unit*)
DDR	*Double data rate*
DMA	*Direct Memory Access*
DM	Mapeador de dispositivos (*Device Mapper*)
DRAM	Memória Dinâmica de Acesso Aleatório (*Dynamic Random-Access Memory*)
FAQ	Perguntas Frequentes (*Frequently Asked Questions*)
FDE	Criptografia do disco inteiro (*Full Disk Encryption*)
GDB	*GNU Debugger*
GNU	Acrônimo recursivo para *GNU is Not Unix*
GRUB	*GRand Unified Bootloader*
HDD	Disco Rígido (*Hard Disk Drive*)
IOMMU	*Input–Output Memory Management Unit*
I/O ou E/S	Entrada/Saída (*Input/Output*)
LUKS	*Linux Unified Key Setup*
LVM	Gerenciador de Volume Lógico (*Logical Volume Manager*)
MBR	*Master Boot Record*
MV ou VM	Máquina Virtual (*Virtual Machine*)
PBA	*Pre-Boot Authentication*[1]
PCI	*Peripheral Component Interconnect*
PSID	*Physical Presence Security Identifier*
RAID	*Redundant Array of Inexpensive/Independent Drives*
RAM	Memória de Acesso Aleatório (*Random Access Memory*)
RGB	Vermelho, Verde, Azul (*Red, Green, Blue*)

RSA	Iniciais dos criadores do algoritmo de criptografia: Rivest, Shamir e Adleman
SED	*Self-Encrypting drive*
SHA	*Secure Hash Algorithm*
SID	*Security Identifier*
SO ou OS	Sistema Operacional (*Operating System*)
SRAM	Memória Estática de Acesso Aleatório (*Static Random-Access Memory*)
SSD	Unidade de Estado Sólido (*Solid-state drive*)
SSH	*Secure Shell*
TCG	*Trusted Computing Group*
UEFI	*Unified Extensible Firmware Interface*
VMX ou VT-x	*Virtual Machine Extensions*
VT-d	*Virtualization Technology for Directed I/O*
XTS	*Ciphertext Stealing*

[1]Apesar de ter sido utilizada erroneamente como *Pre-Boot Authorization* pelo *linuxpba* na seção 3.2.1, o significado correto é Pre-Boot Authentication, como pode ser visto na especificação OPAL 2.00

Capítulo 1

Avisos legais

O presente trabalho trata do estudo de criptografia e algumas de suas vulnerabilidades já conhecidas. Tais vulnerabilidades não devem ser utilizadas para propósitos ilegais, tanto em território brasileiro, quanto em ambiente internacional. Os autores não se responsabilizam por qualquer uso indevido desse estudo.

O trabalho não apresenta nenhuma vulnerabilidade de dia-zero (*Zero-day vulnerability*), portanto não houve a necessidade de apresentá-las com antecedência para as empresas envolvidas.

Por fim, os autores reforçam que fica a cargo do leitor a responsabilidade sobre qualquer eventual falha, perda de dados, quebra de equipamento ou qualquer outra possível complicação que possa ocorrer. O leitor também é responsável por aprender o conhecimento técnico necessário para reproduzir qualquer metodologia apresentada neste trabalho. É recomendado o *backup* de todos os dados anteriormente a qualquer tentativa de criptografia do seu disco, seja ele um HDD ou um SSD.

Capítulo 2

Introdução

O problema de segurança da informação tem acompanhado a história da humanidade desde épocas remotas como a de Heródoto (c. 484 - c. 425 BC) até a atualidade vide o vazamento liberado por Julian Assange de informações militares sigilosas do governo estadunidense culminando no projeto WikiLeaks (2006) wik e a divulgação dada por Edward Snowden de espionagem global por conta novamente do governo dos Estados Unidos da América.

Remontando à época de Heródoto, apesar de rudimentar a técnica usada por Histieu, seu método se provou eficaz pois o objetivo de enviar uma mensagem sem que a mesma fosse interceptada ou lida foi atingido. Heródoto relata que "para Histieu, quando este estava ansioso para entregar a Aritágoras ordens de revolta, somente encontrou um método seguro, pois as vias estavam vigiadas, para o qual pudesse fazer com que seus desejos fossem satisfeitos: raspar as cabeças de seus escravos mais fiéis e, somente assim, gravar-lhes as letras desejadas e esperar que nascesse novamente. Assim Histieu o fez e tão logo que nasceram os cabelos novamente, enviou seus escravos para a cidade de Mileto ordenando não mais que isto: 'Quando vos encontrardes em Mileto, ordenai Aristágoras que vos raspe as cabeças e então que as olhe em seguida'". Herodotus (440 BC)

Outro exemplo icônico na história da humanidade sobre segurança da informação foi relatado num longa lançado em 2014, *O jogo da imitação* imi. Nesse episódio, a Alemanha criara um dispositivo o qual cifrava todas as mensagens enviadas a suas tropas e, em seguida, cada tropa possuindo a chave para reverter o processo, podia então ler a mensagem na íntegra.

A enigma se utilizou de uma técnica similar a grupos cíclicos e da cifra de César, sendo assim um método de segurança distinto que Histieu usara em seu tempo; Histieu usara esteganografia [1] e a Alemanha, criptografia. [2]

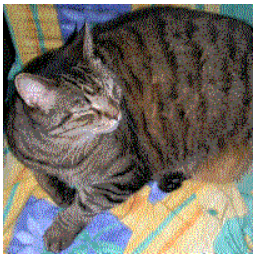
Apesar de *kryptós* e *steganós* serem muito similares em seus significados, esses conceitos desenvolvem papéis claramente distintos no contexto de segurança da informação. A esteganografia se preocupa em esconder a mensagem original seja em um arquivo de áudio, vídeo ou imagem, por meio de interferência de sinal físico ou algum outro como um quebra cabeça. Essa técnica não se preocupa em utilizar cifras e a mensagem pode simplesmente estar presente mas escondida em texto puro. A criptografia por sua vez, se preocupa somente em cifrar a mensagem desejada, possibilitando que qualquer pessoa que não possua a chave possa olhar o texto cifrado e não possa recuperar o texto original.

Um exemplo de esteganografia pode ser visualizado na imagem 2.1. À esquerda, temos uma imagem de uma árvore. Porém, se extrairmos somente os dois bits menos significativos

[1] Do grego steganós [$\sigma\tau\epsilon\gamma\alpha\nu\acute{o}\varsigma$], significando escondido e graphein [$\gamma\rho\acute{\alpha}\phi\epsilon\iota\nu$], verbo escrever.

[2] Do grego kryptós [$\kappa\rho\upsilon\pi\tau\acute{o}\varsigma$], significando secreto e graphein [$\gamma\rho\acute{\alpha}\phi\epsilon\iota\nu$], verbo escrever.

(a) *Imagem com figura escondida*



(b) *Figura original*

Figura 2.1: *Exemplo de esteganografia*

de cada componente RGB e normalizarmos seus valores, teremos a figura à direita.

Para criptografia, temos o seguinte exemplo 1 usando a cifra Camellia. [3]

Listing 1: *Exemplo criptografia*

```
#!/bin/bash
cd /tmp
echo "exemplo de criptografia" | openssl \
  camellia-128-cbc -base64 > exemplo.enc # senha 'exemplo'
cat exemplo.enc
# saida:
# 'U2FsdGVkX1+rTW9XRCvqtrDPRHB0cKUavB78dWzJTFQ0+f4/hCIa+
#  vFDvOR6k7Ld'
cat exemplo.enc | openssl \
  camellia-128-cbc -base64 -d # senha 'exemplo'
# saida: "exemplo de criptografia"
exit 0
```

Embora usemos ambos os métodos para sigilo atualmente, o presente trabalho manifestará interesse somente em criptografia para armazenamento de informação em HDD e SSD.

2.1 Motivações e objetivos

HHDs (Hard Disk Drive) e SSDs (Solid State Drive) são ambos dispositivos de armazenamento em massa amplamente utilizados na maioria das máquinas atuais. Neles armazenamos nossos arquivos pessoais quase que em sua totalidade bem como instalações de Sistemas Operacionais, programas como navegadores e editores de texto ou e-mail e chaves de autenticação de *login* em servidores.

Esses dispositivos têm como principais diferenças preço e taxa de transferência, do ponto de vista do usuário final, e tecnologia de armazenamento, sob a perspectiva de arquitetura.

HHDs trabalham com estado magnético internamente, são mais lentos porém têm custo por capacidade consideravelmente reduzido quando comparados com SSDs, que trabalham com memórias *flash*. Enquanto um HDD pode chegar a taxas de 227MB/s (leitura) e 186MB/s (escrita)[4] custando US$0.06/GB[5], um SSD pode atingir 3.500MB/s (leitura) e

[3] Caso o leitor execute este código, é esperado que sua saída de exemplo.enc seja diferente.

[4] Em um Seagate Barracuda PRO http://www.expertreviews.co.uk/internal-hard-drives/1406336/best-internal-hard-drives-the-best-high-capacity-hdds-to-buy-from

[5] https://www.amazon.com/Seagate-BarraCuda-3-5-Inch-Internal-ST6000DM004/dp/B01LOOJBH8

2.100MB/s (escrita)[6] custando US0.58/GB[7].

Segundo Kipp (2015), os próximos anos ainda compartilharão de máquinas com HDDs e SSDs.

Tendo em vista a importância de tais dispositivos, a necessidade de assegurar sigilo das informações neles contidas, o difundido uso do módulo *cryptsetup* e a recente especificação OPAL pela TCG (*Trusted Computing Group*) - ambos técnicas de criptografia, é necessária uma revisão sobre a eficácia de proteção e análise de desempenho dessas técnicas nos dispositivos mencionados.

Os objetivos deste trabalho são:

1. Entender o uso das ferramentas *cryptsetup* - com gerência de senhas pelo LUKS *Linux Unified Key Setup* - e OPAL, utilizando como interface o projeto SEDutil.

2. Fazer uma análise de desempenho dessas ferramentas levando em consideração suportes especiais por parte da CPU (Unidade Central de Processamento).

3. Apresentar vulnerabilidades que cada ferramenta se encontra exposta.

4. Apresentar métodos para diminuir ataques a essas vulnerabilidades.

5. Demonstrar que ambas podem ser vulneráveis quando a máquina estiver ligada ou desligada mas o esforço necessário aumenta para extração das informações dependendo do cenário.[8]

2.2 Desenvolvimento

O desenvolvimento de nosso trabalho foi organizado da seguinte maneira. Primeiramente estudamos as ferramentas *cryptsetup* e *LUKS*. Para tal, foi necessária a investigação de seu comportamento junto ao *Kernel* do *Linux*.

Deu-se então o estudo do módulo de mapeamento de dispositivos (*dm_ mapper*), em seguida o módulo de mapeamento de criptografia (*dm_ crypto*) e o módulo de criptografia (XTS), para contemplar o caso em que o processador não provê instruções AES. Após o entendimento de sua interação com o Sistema Operacional, lemos o manual de instruções do *cryptsetup* para criptografar um dispositivo e posteriormente obter análise de desempenho.

Para o SEDutil, buscamos conhecimento intra dispositivo, uma vez que a criptografia é dada no seu interior e é totalmente transparente para a máquina e Sistema Operacional. De forma análoga, foi estudado como usar a ferramenta para poder criptografar um dispositivo e obter análise de desempenho.

Posteriormente, foram feitas as análises de desempenho a fim de se ter conhecimento, na máquina testada em questão, qual seria a diferença da taxa de transmissão em ambos os casos - seguro e inseguro.

Por fim, foram buscadas na literatura quais são algumas das formas de ataque para as duas técnicas e como podemos diminuir os riscos de se ter os dados expostos. Também fez-se necessário o estudo do contexto de cada tipo de ataque.

[6] Em um SSD Samsung 960 PRO http://www.samsung.com/semiconductor/minisite/ssd/product/consumer/960pro.html

[7] https://www.amazon.com/Samsung-960-PRO-Internal-MZ-V6P512BW/dp/B01LXS4TYB

[8] Apesar de não ser o foco do trabalho, este pode ser um primeiro contato com métodos de extração de informação para interessados em computação forense.

Capítulo 3

Full Disk Encryption

Neste capítulo detalharemos como dá-se a criptografia dos dados armazenados em memórias não voláteis do tipo HDD e SSD, incluindo suas interações com o Sistema Operacional quando for o caso e como solicitar que o S.O. ou dispositivo comece a utilizar um dos métodos de proteção.

O ato de criptografar um dispositivo inteiro do tipo HDD ou SSD chamaremos de *Full Disk Encryption* ou simplesmente FDE.

3.1 Cryptsetup e Linux Unified Key Setup (LUKS)

3.1.1 Device Mapper (DM)

Device Mapper é um driver do *kernel* cuja finalidade é prover um framework para gerenciamento de volumes através de um modo genérico para os dispositivos mapeados e serem usados como volumes lógicos. Esse *driver* não tem conhecimento sobre grupo de volumes metadados de formatação. Ademais, o *Device Mapper* provê o cerne para inúmeras tecnologias de níveis mais altos.

Somado ao gerenciador de volume lógico (LVM), o *Device Mapper* também disponibiliza uma interface de comunicação com o *dm-multipath* e *dmraid* entre outros os quais fogem do escopo desse estudo. Porém, o de importância para o presente trabalho é o *dm-crypt*. Todas as chamadas para o *Device Mapper* são dadas pelo comando *dmsetup*.

A seguir temos ilustrações de onde precisamente residem os drivers *dm* e *dm-crypt*. A figura 3.1 refere-se à versão 4.0 do *kernel* e a figura 3.2 refere-se à versão 3.17.

No capítulo DM_Crypt, abordaremos como a criptografia de blocos é dada usando a transparência provida pelo *Device Mapper*.

3.1.2 DM_Crypt

Utilizando-se do *driver device mapper*, o alvo do *dm-crypt* (*device mapper crypt*) provê criptografia de dispositivos bloco transparente usando a API de criptografia do *kernel* (*kernel crypto API*). Basicamente o usuário especifica uma cifra simétrica, um modo de criptografia, uma chave de tamanho suportado pelo *kernel*, um modo de geração com valor inicial (IV) e então é criado um novo dispositivo no diretório /dev.

Escritas nesses dispositivos serão criptografadas e leituras serão decriptografadas utilizando o método escolhido. A cargo do usuário, ele pode montar o sistema de arquivo como usualmente ou através da pilha *dm-crypt* com outro dispositivo como RAID ou LVM.

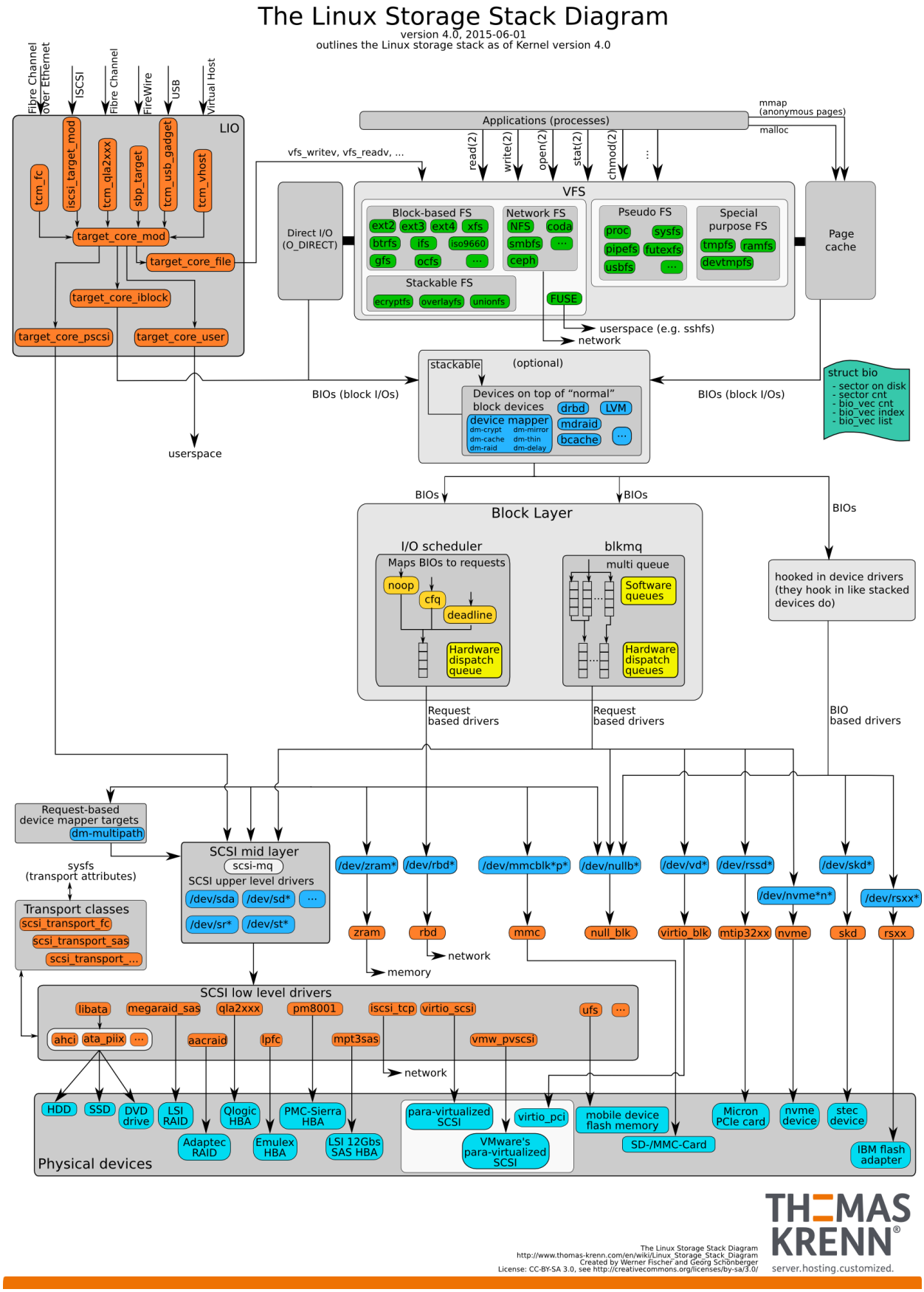


Figura 3.1: *Linux storage stack diagram v4.0*

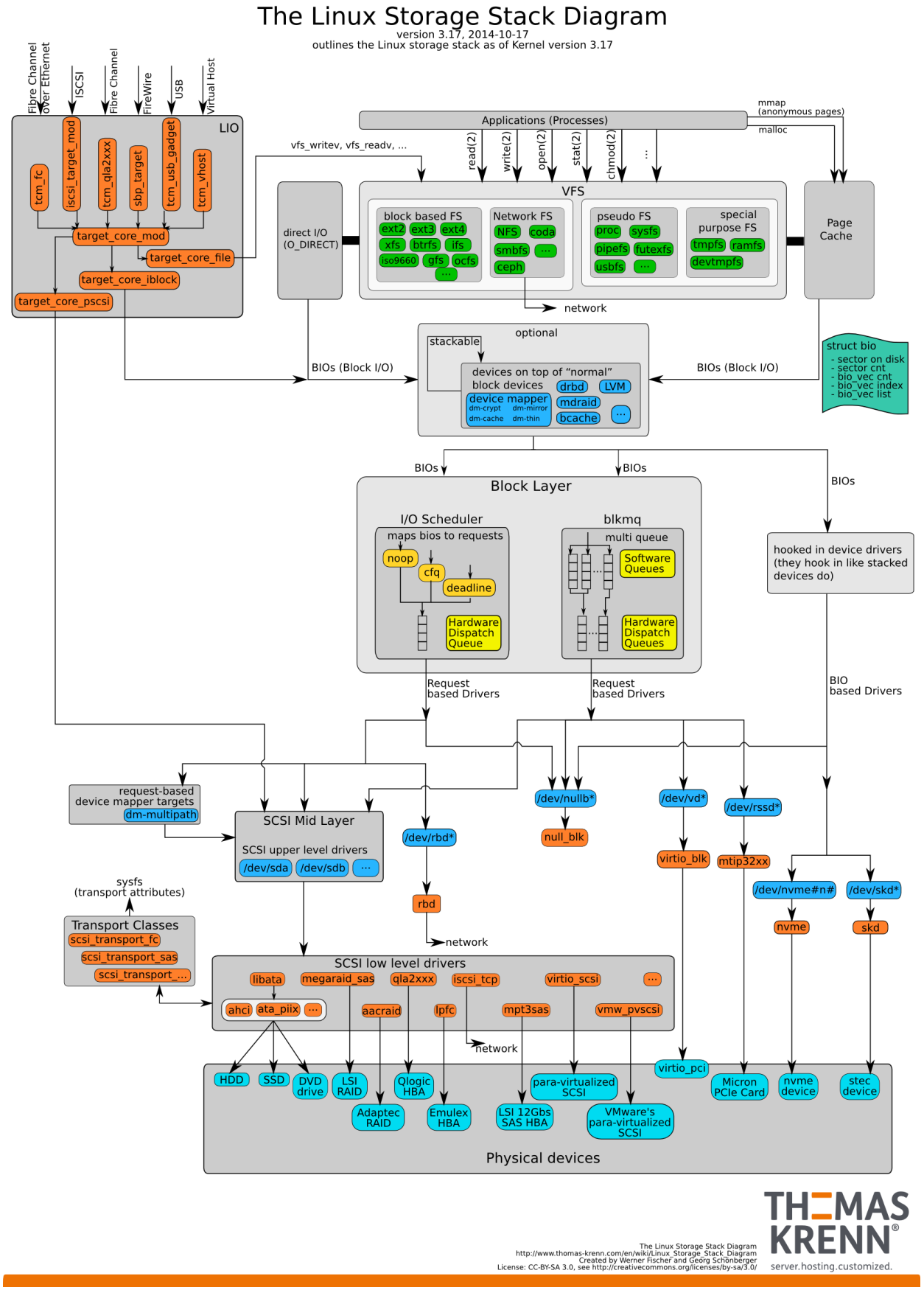


Figura 3.2: *Linux storage stack diagram v3.17*

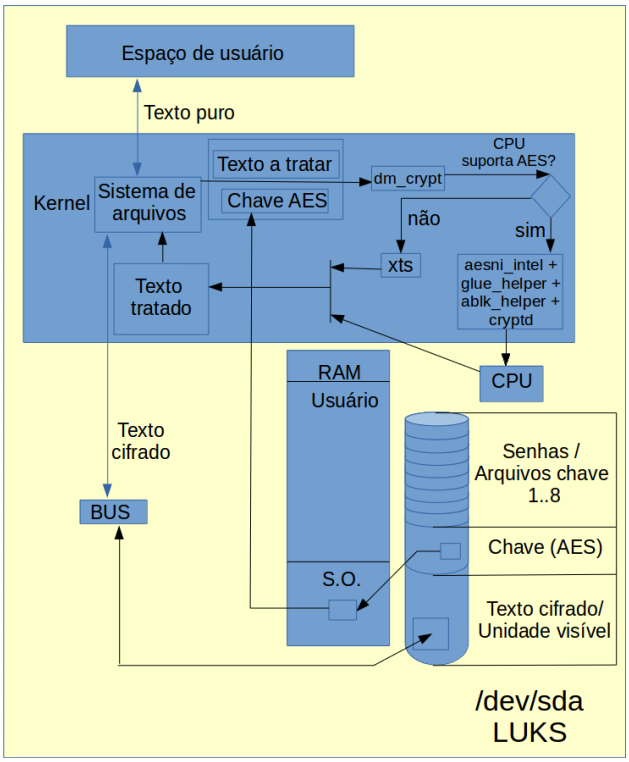


Figura 3.3: *Fluxo de dados simplificado usando sistema LUKS*

Consultas de cifras e modos suportados pelo *kernel* são fornecidas através do arquivo /proc/crypto, porém manifestamos interesse no XTS AES PLAIN 64 com chaves de 128, 192 e 256 bits. Vale notar que /proc/crypto disponibiliza somente os módulos carregados em tempo de execução, sendo assim possível adicionar ou remover modos suportados.

Tendo escolhido a cifra e modo, *dm-crypt* se encapsula dentro do dm e toda requisição é então transparente ao usuário após inicialização.

3.1.3 LUKS - Linux Unified Key Setup

O sistema LUKS é a forma padronizada de criptografia de HDD para o Sistema Operacional GNU/Linux. Providenciando um padrão em disco, ele facilita a compatibilidade entre as muitas distribuições além de disponibilizar um gerenciamento seguro de chaves ou senhas quando usado por múltiplos usuários.

Esse método gerencia todas as informações em um cabeçalho. Usando o sistema XTS AES, temos a configuração do LUKS na seguinte disposição: uma chave AES, segurada por alguma senha ou arquivo chave que pode conter até 8MB, oito entradas para as senhas de usuário e internamente regidos por algum sistema de arquivo como *ext4*. A chave AES não pode em hipótese alguma ser trocada; uma vez reescrita, todos os dados dentro da partição são perdidos pois é exatamente ela que é usada para tratar os dados. Porém qualquer uma das 8 chaves pode ser facilmente revogada, trocada, excluída ou inserida desde que uma permaneça para acesso aos dados e chave AES.

Na figura 3.3 temos uma simplificação de como os dados são tratados usando o sistema LUKS com e sem o suporte da CPU.

3.1.4 Como utilizar o LUKS

Os passos para criptografar e utilizar um disco utilizando o LUKS são os seguintes:

- Certificar que o módulo *dm_ crypt* do *kernel* está habilitado
- Certificar que o utilitário de linha de comando *cryptsetup* está instalado. É ele que faz a interface com o módulo do *kernel*.
- Preparar o disco para a utilização de criptografia, fornecendo senha e métodos de criptografia a serem utilizados e verificar que o dispositivo está pronto.
- Montar e desmontar o dispositivo criptografado.

Aqui usaremos o dispositivo */dev/sdb* e o nome *sagitta* para o *Device Mapper*

3.1.4.1 Certificando que o módulo *dm_ crypt* do *kernel* está habilitado

Podemos usar o seguinte comando para checar se o módulo está habilitado:

```
# lsmod | grep dm_crypt
```

Uma saída não-vazia indica que o módulo está carregado.

3.1.4.2 Certificando que o utilitário de linha de comando *cryptsetup* está instalado.

O comando a seguir retorna a versão do *cryptsetup*. Caso ele não esteja instalado, instale o pacote *cryptsetup* disponível para a sua distribuição.

```
# cryptsetup --version
```

3.1.4.3 Preparando o disco para a utilização do LUKS

O comando a seguir prepara o dispositivo para a utilização do LUKS. Ao contrário do que o comando *luksFormat* dá a entender, ele apenas prepara o cabeçalho LUKS com as informações necessárias para o funcionamento da criptografia ao invés de formatar o dispositivo para algum sistema de arquivos. As opções serão descritas a seguir.

```
# cryptsetup --verbose --cipher aes-xts-plain64 --key-size 256 --hash sha256
  --iter-time 2000 --use-random --verify-passphrase luksFormat /dev/sdb
```

- **–verbose (ou -v):** Verboso. Mostra mais informações durante a execução.
- **–cypher (ou -c):** Escolhe a cifra a ser utilizada, no caso: *aes-xts-plain64*.
- **–key-size (ou -s):** Tamanho da chave em bits. Precisa ser um múltiplo de 8.
- **–hash (ou -h):** Especifica o hash a ser utilizado, no caso: *sha256*.
- **–iter-time (ou -i):** O número de milisegundos a serem gastos com o processamento da senha.
- **–use-urandom (ou, alternativamente –use-random):** Define se o sistema deverá utilizar o /dev/random ou o /dev/urandom para criar a chave mestra.
- **–verify-passphrase (ou -y):** Pede por uma confirmação da senha escolhida.
- **luksFormat /dev/sdb:** Utiliza o comando de criação do cabeçalho LUKS no dispositivo escolhido.

3.1.4.4 Verificando que o disco está preparado

Para fazer a verificação, podemos utilizar o comando *luksDump* no dispositivo. A seguir, um exemplo de saída do comando:

```
# cryptsetup luksDump /dev/sdb
```

```
LUKS header information for /dev/sdb

Version:        1
Cipher name:    aes
Cipher mode:    xts-plain64
Hash spec:      sha256
Payload offset: 4096
MK bits:        256
MK digest:      d2 ac f5 e9 0d ff 78 70 aa 81 3b 91 d9 e9 39 0b d1 ab c4 03
MK salt:        96 1a 30 cf ae 93 da f5 92 9f 60 15 b5 7c 08 b0
                c8 f2 13 af 9c 01 f4 ef 7d 00 c1 37 f5 05 0f c8
MK iterations:  75000
UUID:           ac594dc0-cead-4068-9573-796348a75330

Key Slot 0: ENABLED
  Iterations:             301886
  Salt:                   1b ad 27 4b d2 b8 dc 8d f5 b3 62 fa 95 b1 50 4f
                            fa f9 a3 d7 33 ce f2 5d f7 56 4a 1c 8c 92 54 08
  Key material offset:  8
  AF stripes:             4000
Key Slot 1: DISABLED
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
```

3.1.4.5 Montando o dispositivo e adicionando um sistema de arquivos

Primeiramente precisamos abrir o dispositivo, utilizando o *Device Mapper*:

```
# cryptsetup open --type luks device sagitta
```

```
# cryptsetup open --type luks /dev/sdb sagitta
Informe a frase secreta para /dev/sdb:
# ls /dev/mapper/
control  sagitta
```

E, em seguida, criar um sistema de arquivos no dispositivo, utilizando o *Device Mapper*:

```
# mkfs -t ext4 /dev/mapper/sagitta
mke2fs 1.42.9 (4-Feb-2014)
Rótulo do sistema de arquivos=
OS type: Linux
Tamanho do bloco=4096 (log=2)
Tamanho do fragmento=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
237104 inodes, 946864 blocks
47343 blocks (5.00%) reserved for the super user
Primeiro bloco de dados=0
Máximo de blocos de sistema de arquivos=973078528
```

```
29 grupos de blocos
32768 blocos por grupo, 32768 fragmentos por grupo
8176 inodes por grupo
Cópias de segurança de superblocos gravadas em blocos:
  32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: pronto
Gravando tabelas inode: pronto
Creating journal (16384 blocks): concluído
Escrevendo superblocos e informações de contabilidade de sistema de arquivos:
   concluído
```

Agora, o dispositivo /dev/mapper/sagitta pode ser montado como qualquer outro:

```
# mount /dev/mapper/sagitta /mnt/cripto/
```

Quando o dispositivo estiver pronto para ser fechado, devemos desmontá-lo normalmente e depois fechar o LUKS.

```
# umount /mnt/cripto
```

```
# cryptsetup close sagitta
```

Esse último passo fecha o LUKS e limpa a chave da memória do *kernel*.

3.1.5 Onde utilizar o LUKS

O LUKS pode ser utilizado em qualquer dispositivo ou partição. Por exemplo, na seção anterior, poderíamos ter usado */dev/sdb1* com o mesmo efeito, possibilitando até a existência de partições criptografadas e não-criptografadas no mesmo dispositivo físico.

Além disso, muitas distribuições modernas já disponibilizam a opção de criptografia na própria interface gráfica do instalador. No caso em que é feita essa escolha, é importante lembrar que a área de *swap* também deve ser criptografada, afinal ela pode vir a armazenar qualquer informação de senhas e chaves criptográficas. Para tal, precisamos colocar a área de *swap* na tabela /etc/crypttab.

```
swap  /dev/<partição>  /dev/urandom  swap,noearly
```

e alterar sua entrada na tabela /etc/fstab:

```
/dev/mapper/swap none swap sw 0 0
```

3.1.6 LUKS em SSDs

A seção 5.19 do FAQ do cryptsetup faz um alerta contra a utilização do LUKS em SSDs e *Flash Drives*.

A ressalva diz respeito à maneira como os SSDs fazem a sobrescrita de dados. Dependendo do fabricante e do funcionamento interno do dispositivo, a sobrescrita em um mesmo endereço não é garantida, acarretando na possível perda de algumas propriedades do LUKS como, por exemplo, a revogação, troca ou exclusão das chaves. Assim, é recomendado que o seu modelo de atacante seja revisto. Se há a possibilidade que senhas antigas sejam comprometidas, então o uso do LUKS em SSDs não deve ser feito. Afinal, devido ao modo como são feitas as escritas no SSD, os cabeçalhos antigos podem ainda estar no dispositivo, possibilitando o acesso aos dados, mesmo com uma chave já revogada.

Liu *et al.* (2017) e Choi *et al.* (2014) suportam as ressalvas com propostas de, além do uso de criptografia, garantia de remoção verdadeira dos dados quando requisitados.

3.2 SEDutil

Como descrito no capítulo 2.1, SSDs e HDDs são ambos dispositivos de armazenamento secundário mas com tecnologias distintas para tal. Visando segurança dos dados armazenados em SSD, a *Drive Trust Alliance*[1] tem um projeto colaborativo para SSDs que suportam SED, o SEDutil[2].

A premissa para que seu projeto seja executado com sucesso no dispositivo alvo é que este esteja em conformidade com as especificações dadas pela Trusted Computing Group, Incorporated (2012), OPAL v2.

O projeto SEDutil tem como objetivo uma implementação livre de FDE usando SED em dispositivos compatíveis com a especificação OPAL 2.0 ou superior descrita pelo órgão *Trusted Computing Group* (TCG), órgão esse que se preocupa em criar padrões abertos e livres para segurança de dados.

A OPAL propõe que fabricantes de armazenamento como Samsung, Intel e OCZ implementem uma API em seus dispositivos para que, quando chamados de forma correta, o dispositivo tenha uma *Master Boot Record* (MBR) falsa para que no momento de *boot* da máquina, a BIOS ou UEFI consiga carregá-la e então com uma imagem mínima de S.O. como *Linux Pre Boot Authentication* (linuxPBA) é trazida à memória e assim é pedido ao usuário que insira a senha do dispositivo. Quando a senha correta é inserida, o dispositivo é então liberado e disponibiliza realmente suas partições internas e MBR ou UEFI caso presente.

Dispositivos que suportam OPAL tem como responsabilidade criptografar e decriptografar todos os dados armazenados internamente regidos por AES-128, AES-192 ou AES-256, a escolha do fabricante e com algum tipo de *hash* para confronto das senhas, tornando toda operação de segurança transparente para usuários e S.O.s.

Desse modo, como a segurança é feita no próprio dispositivo, a CPU fica totalmente livre de tal encargo e enquanto os processos esperam pelos dados, a CPU pode continuar outras tarefas escalonadas até que receba uma interrupção de E/S.

Tendo entendido o funcionamento e a especificação da OPAL, o binário SEDutil tem como responsabilidade se comunicar de forma adequada com a API disponível para que corretamente configure as travas e a partição escondida de MBR ou UEFI.

Abaixo temos uma ilustração de como os dados são lidos ou escritos no dispositivo e como é dada sua criptografia ou decriptografia de acordo com a operação desejada.

Na figura 3.4, temos uma ilustração de como os dados tratados são enviados e recebidos usando o sistema SEDutil.

[1] https://www.drivetrust.com/
[2] https://github.com/Drive-Trust-Alliance/sedutil

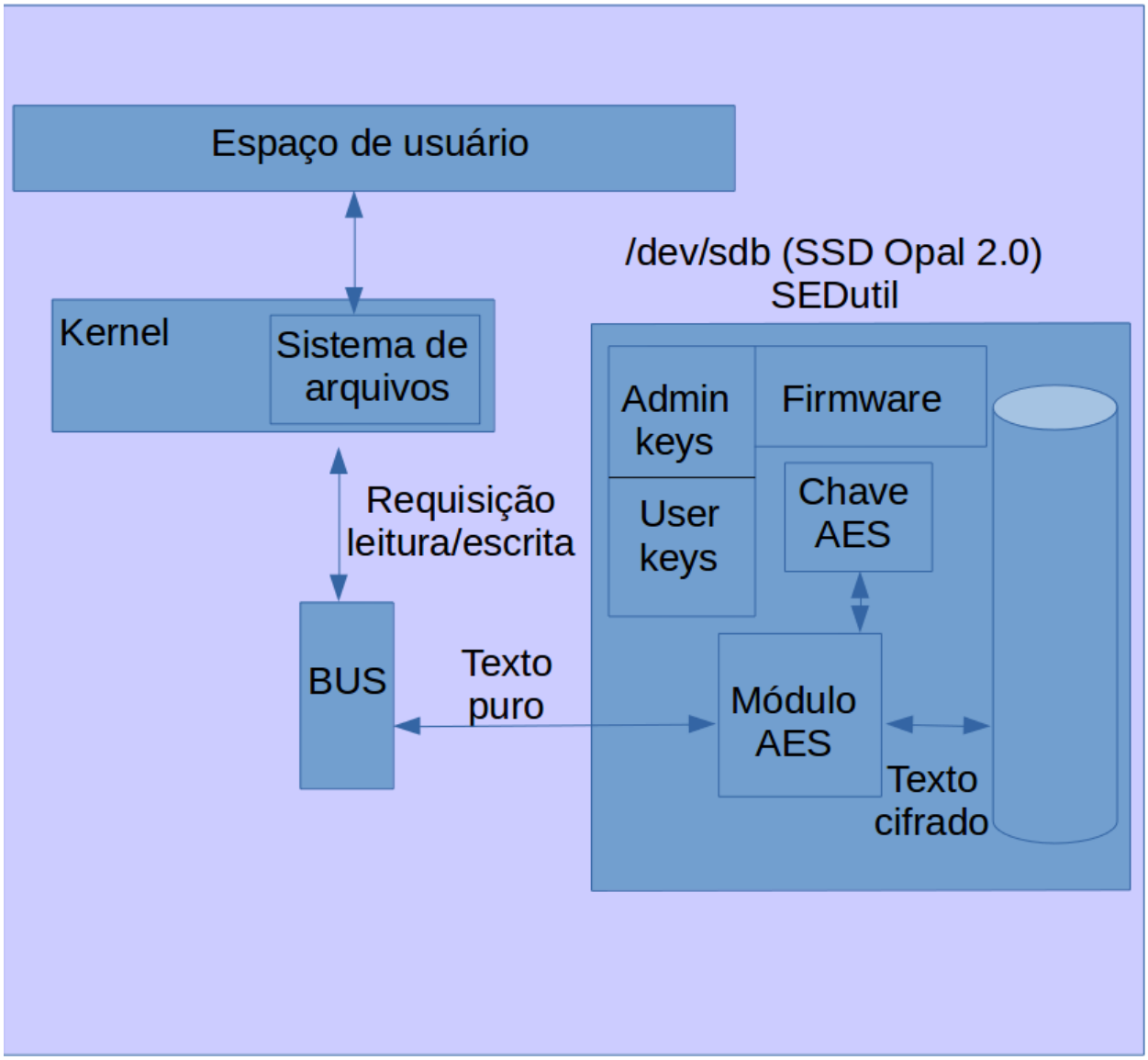


Figura 3.4: *Fluxo de dados pelo SEDutil*

Nas sessões seguintes iremos trazer um passo-a-passo de como habilitar a trava OPAL 2.x em seu dispositivo compatível. Quando todos os passos terminarem com sucesso, sempre que a máquina for ligada (não se aplica para reiniciada), será pedida a senha para desbloqueio do mesmo e ela será reiniciada novamente com as verdadeiras partições habilitadas para leitura e escrita.

Todos as instruções foram retiradas do projeto original https://github.com/Drive-Trust-Alliance/sedutil/wiki/Encrypting-your-drive e, caso os dados aqui presentes estejam desatualizados, recomendamos que consulte o *site* citada.

Após esse guia, entraremos em mais detalhes sobre cada comando executado na perspectiva da própria especificação dada pela TCG.

3.2.1 Ativando a trava OPAL

1. SSD alvo nesse tutorial está em */dev/sd**b***
2. *pendrive* alvo está em */dev/sd**e***. Os caminhos em /dev/sdX são distintos em cada máquina.

3. Alguns comandos podem causar perdas irreversíveis de dados. Sempre certifique-se que o comando a ser executado tenha como alvo o dispositivo desejado.

4. Tomamos também como imagem de recuperação (*Rescue system*) a BIOS e não a UEFI.

5. Como recomendado pelo projeto, foi usado um teclado **us_english**. Caso não disponha de tal mapeamento, **USE** teclado **QWERTY** e somente caracteres **ASCII**.

Para habilitar a trava, execute as seguintes instruções na ordem que segue:

1. Baixe, descomprima a imagem de recuperação e escreva em um *pendrive*.

```
# Como root, execute
mkdir /tmp/sed_bios
cd /tmp/sed_bios
wget -c https://github.com/Drive-Trust-Alliance/exec/blob/master/RESCUE32
    .img.gz?raw=true -O RESCUE32.img.gz
gunzip RESCUE32.img.gz
dd if=RESCUE32.img.gz of=/dev/sde
```

2. Reinicie a máquina e inicialize pelo *pendrive*. Todos os comandos doravante deverão ser executados no *prompt* do sistema de recuperação. Após ter reiniciado a máquina e ter carregado o sistema de recuperação, deverá ser apresentada uma tela como a seguinte:

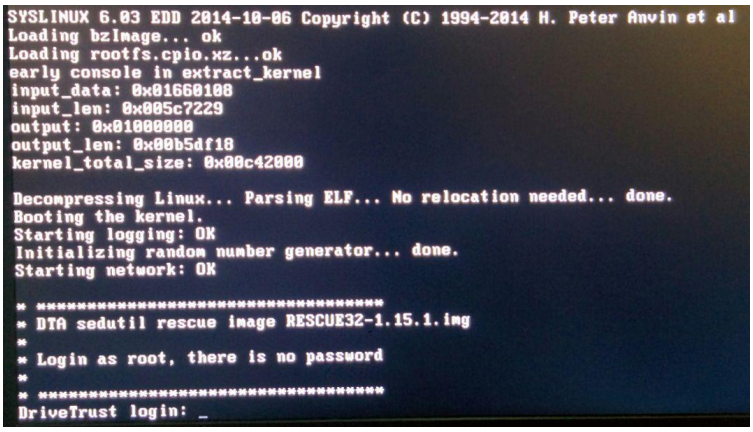


Figura 3.5: *Boot inicial*

3. Digite **root** como *login* (não será pedida senha alguma). A saída do comando a seguir lista quais dispositivos são compatíveis com OPAL 1, OPAL 2 e incompatíveis. Somente podemos prosseguir com dispositivos que suportam a versão 2.

Cada linha apresentadas pelo sedutil-cli --scan apresenta as seguintes informações:

```
/dev/<dispositivo> <suportes> <nome_modelo> <serial>
```

Dispositivos não suportados serão listado como

```
"/dev/<dispositivo>  No ..."
```

Dispositivos que suportam somente a versão 1 serão listados como

```
"/dev/<dispositivo>   1 ..."
```

Dispositivos que suportam somente a versão 2 serão listados como

```
"/dev/<dispositivo>   2 ..."
```

Dispositivos que suportam as versões 1 e 2 serão listados como

```
"/dev/<dispositivo>  12 ..."
```

```
root
sedutil-cli --scan
> Scanning for Opal compliant disks
> /dev/nvme0   2  Samsung SSD 960 EVO 250GB      2B7QCXE7
> /dev/sda    12  Samsung SSD 850 EVO 500GB      EMT01B6Q
> /dev/sdb     2  Crucial_CT250MX200SSD1         MU04
> No more disks present ending scan
```

4. Agora teste o *Pre Boot Authentication* (PBA). Quando solicitada a senha, digite **debug**. Caso digite algo diferente de debug, a máquina será reiniciada.

```
linuxpba
> DTA LINUX Pre Boot Authorization
>
>
> Please enter pass-phrase to unlock OPAL drives: *****
> Scanning....
```

Tabela 3.1: *Saída do Scanning...*

Dispositivo	Nome e Modelo	is OPAL/not OPAL	NOT LOCKED/ Unlocked
Drive /dev/nvme0	Samsung SSD 960 EVO 250GB	is OPAL	NOT LOCKED
Drive /dev/sda	Crucial_CT250MX200SSD1	is OPAL	NOT LOCKED
Drive /dev/sdb	Samsung SSD 850 EVO 500GB	is OPAL	NOT LOCKED

Certifique-se que o dispositivo escolhido, sdb no nosso exemplo, tenha saída **is OPAL**

5. Habilite a trava e copie PBA para a MBR que será exposta quando o dispositivo não estiver destravado.

```
sedutil-cli --initialsetup debug /dev/sdb
sedutil-cli --enablelockingrange 0 debug /dev/sdb
sedutil-cli --setlockingrange 0 lk debug /dev/sdb
sedutil-cli --setmbrdone off debug /dev/sdb
cd /tmp
cp /usr/sedutil/BIOS32-1.15.1.img.gz . # << 1.15.1 pode variar!
gunzip BIOS32-1.15.1.img.gz
sedutil-cli --loadpbaimage debug BIOS32-1.15.1.img /dev/sdb
```

Compare com as saídas esperadas:

Tabela 3.2: *Saída do Scanning... após dispositivo ser travado com sucesso.*

Dispositivo	Nome e Modelo	is OPAL/not OPAL	NOT LOCKED/ Unlocked
Drive /dev/nvme0	Samsung SSD 960 EVO 250GB	is OPAL	NOT LOCKED
Drive /dev/sda	Crucial_CT250MX200SSD1	is OPAL	NOT LOCKED
Drive /dev/sdb	Samsung SSD 850 EVO 500GB	is OPAL	Unlocked

```
sedutil-cli --initialsetup debug /dev/sdb
> - 14:06:39.709 INFO: takeOwnership complete
> - 14:06:41.703 INFO: Locking SP Activate Complete
> - 14:06:42.317 INFO: LockingRange0 disabled
> - 14:06:42.694 INFO: LockingRange0 set to RW
> - 14:06:43.171 INFO: MBRDone set on
> - 14:06:43.515 INFO: MBRDone set on
> - 14:06:43.904 INFO: MBREnable set on
> - 14:06:43.904 INFO: Initial setup of TPer complete on /dev/sdb
sedutil-cli --enablelockingrange 0 debug /dev/sdb
> - 14:07:24.914 INFO: LockingRange0 enabled ReadLocking,WriteLocking
sedutil-cli --setlockingrange 0 lk debug /dev/sdb
> - 14:07:46.728 INFO: LockingRange0 set to LK
sedutil-cli --setmbrdone off debug /dev/sdb
> - 14:08:21.999 INFO: MBRDone set off
cd /tmp
cp /usr/sedutil/BIOS32-1.15.1.img.gz .
gunzip BIOS32-1.15.1.img.gz
sedutil-cli --loadpbaimage debug BIOS32-1.15.1.img /dev/sdb
> - 14:10:55.328 INFO: Writing PBA to /dev/sdb
> 33554432 of 33554432 100% blk=1500
>  14:14:04.499 INFO: PBA image BIOS32-1.15.1.img written to /dev/sdb
```

6. Após esse passo, verifique que o dispositivo foi travado com sucesso. Quando pedida a senha, entre com "debug"(sem aspas).

```
linuxpba
>
> DTA LINUX Pre Boot Authorization
>
>
> Please enter pass-phrase to unlock OPAL drives: *****
> Scanning....
```

Caso tenha obtido *Unlocked* como saída para o dispositivo alvo, a trava se deu com sucesso. Caso contrário, é necessário reverter o processo desabilitando a trava incompleta 3.2.2 ou revertendo ao estado de fábrica 3.2.4.

Até agora temos: 1) PBA carregada na MBR falsa. 2) Dispositivo travado com senhas sendo "debug". Iremos agora configurar o dispositivo para que utilize as senhas desejadas pelo usuário.

7. Configurando senhas desejadas: Senhas do *Security Identifier* (SID), não necessariamente deve ser a mesma de administrador, *Admin1*.

```
sedutil-cli --setsidpassword debug senha_desejada /dev/sdb
sedutil-cli --setadmin1pwd debug senha_admin_desejada /dev/sdb
```

Saída esperada:

```
sedutil-cli --setsidpassword senha_admin_desejada /dev/sdb
sedutil-cli --setadmin1pwd  senha_admin_desejada /dev/sdb
> - 14:20:53.352 INFO: Admin1 password changed
```

Para testar se sua senha foi entrada de forma desejada, execute o seguinte comando para mostrar a MBR escondida:

```
sedutil-cli --setmbrdone on senha_admin_desejada /dev/sdb
```

Saída esperada dos comandos:

```
sedutil-cli --setmbrdone on senha_admin_desejada /dev/sdb
> - 14:22:21.590 INFO: MBRDone set on
```

Neste ponto, o dispositivo se encontra destravado. Para ativar a trava, desligue completamente a máquina.

Caso desligue e ligue a máquina, uma tela similar a figura 3.6 será apresentada ao tentar carregar o SO a partir do SSD que foi travado com sucesso.

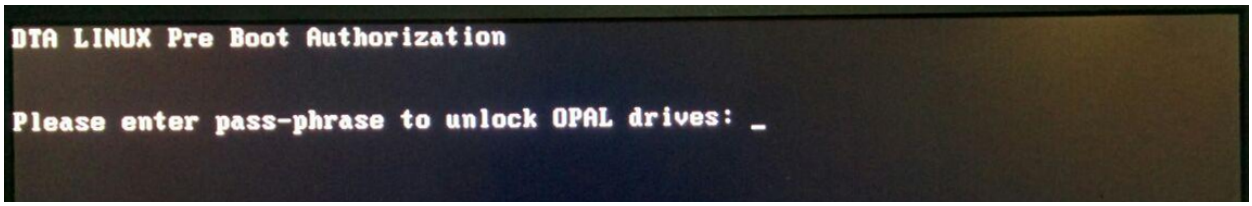


Figura 3.6: *Tela de senha de um dispositivo protegido com OPAL 2 usando SEDutil*

Dados que todos os passos foram executados com sucesso, seu dispositivo está seguro utilizando os protocolos descritos pela especificação TCG OPAL 2.0.

A seguinte sessão descreve como reverter a trava ou, em caso de esquecimento da senha de administrador, restaurar os padrões de fábrica.

3.2.2 Habilitando e desabilitando a trava OPAL

Uma vez configurado seu dispositivo, é possível habilitar e desabilitar a trava de forma mais simples pois o mesmo já está preparado para esconder e abrir a MBR verdadeira.

Para desabilitar a trava, fazendo com que não seja pedida senha quando a máquina for ligada, carregue o sistema de recuperação utilizando o mesmo método descrito no início de 3.2.1 e execute:

```
sedutil-cli --disableLockingRange 0 <senha> <dispositivo>
sedutil-cli --setMBREnable off <senha> <dispositivo>
```

Exemplo de saída esperada:

```
sedutil-cli --disablelockingrange 0 debug /dev/sdb
> - 14:07:24.914 INFO: LockingRange0 disabled
sedutil-cli --setmbrenable off debug /dev/sdb
> - 14:08:21.999 INFO: MBREnable set off
```

Já, caso seja desejado voltar a trava, basta executar

```
sedutil-cli --enableLockingRange 0 <senha> <dispositivo>
sedutil-cli --setMBREnable on <senha> <dispositivo>
```

Exemplo de saída esperada:

```
sedutil-cli --enablelockingrange 0 debug /dev/sdb
> - 14:07:24.914 INFO: LockingRange0 enabled ReadLocking,WriteLocking
sedutil-cli --setmbrenable on debug /dev/sdb
> - 14:08:21.999 INFO: MBREnable set on
```

3.2.3 Removendo o OPAL

Os comandos descritos nessa sub sessão não devem ter como efeito colateral a perda dos dados no dispositivo. O resultado esperado é somente a remoção da trava OPAL fazendo com que a MBR verdadeira sempre esteja exposta quando a máquina for ligada. **Porém**, alguns *firmwares* não se comportam da maneira esperada e reiniciam o dispositivo para o estado de fábrica, causando perda de todos os dados (https://github.com/Drive-Trust-Alliance/sedutil/wiki/Encrypting-your-drive).

Já, caso seja desejado voltar o OPAL, basta executar

```
sedutil-cli --revertnoerase <senha> <dispositivo>
sedutil-cli --reverttper <senha> <dispositivo>
```

Exemplo de saída esperada:

```
sedutil-cli --revertnoerase debug /dev/sdb
> - 14:22:47.060 INFO: Revert LockingSP complete
sedutil-cli --reverttper debug /dev/sdb
> - 14:23:13.968 INFO: revertTper completed successfully
```

3.2.4 Restaurando padrão de fábrica

Atenção: todos os dados no dispositivo alvo serão perdidos caso execute os comandos abaixo. Use com finalidade de estudo ou caso realmente tenha esquecido a senha de administração.

Para restaurar o padrão de fábrica do dispositivo, devemos ter em mãos um identificador de 32 dígitos chamado PSID encontrado impresso no mesmo. Tal identificador se mostra necessário para que ataques remotos não executem as instruções de restauração, fazendo com que todos os dados sejam perdidos. Um exemplo de PSID pode ser encontrado na figura 3.7[3].

No modo de recuperação, execute

```
sedutil-cli --yesIreallywanttoERASEALLmydatausingthePSID <PSID> <dispositivo>
```

Caso esteja presente na saída *INFO: revertTper completed successfully*, o dispositivo se encontra restaurado.

3.2.5 Instalando um Sistema Operacional em um SSD com OPAL ativado

Caso deseje instalar um S.O. em um SSD compatível com OPAL 2.0, habilite sua trava como instruído na sessão anterior, reinicie a máquina (sem desligá-la para que não seja travado) carregando o instalador do S.O. desejado.

[3] figura retirada do tutorial do SEDutil

Figura 3.7: *PSID de um Samsung PRO 850*

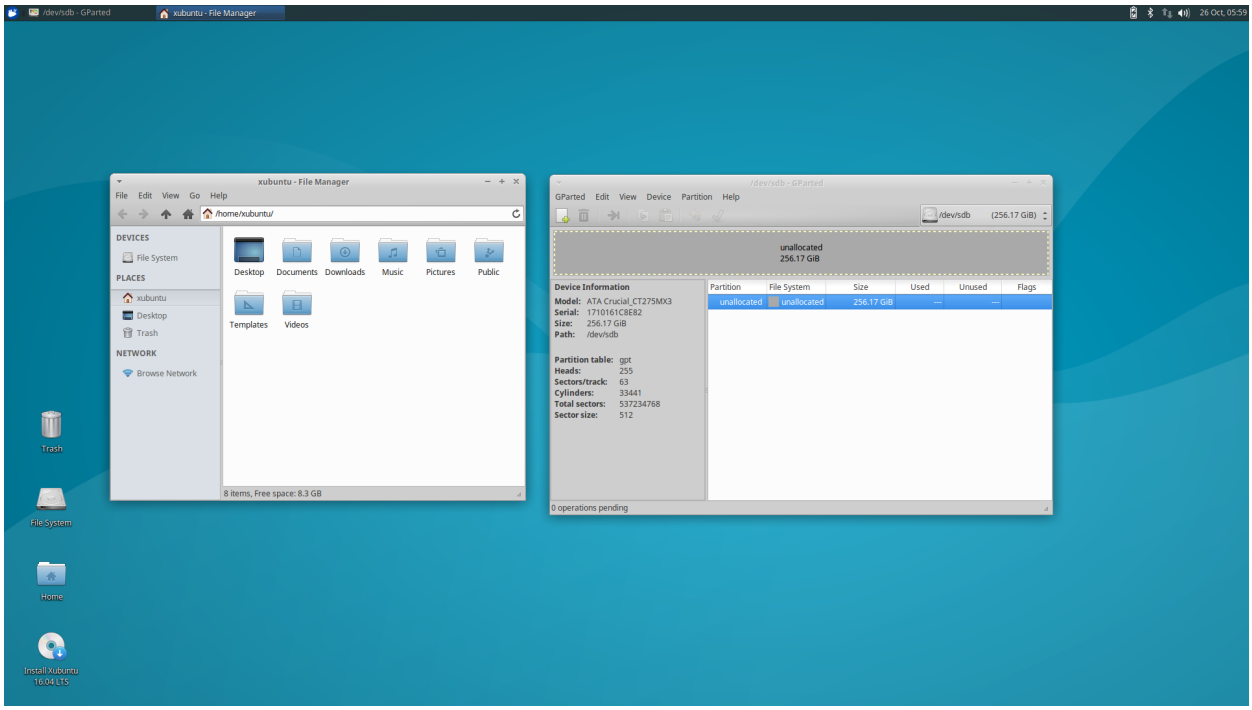
Após esses passos, todo o processo de instalação do Sistema Operacional se dá de forma normal.

Nossos testes com OPAL foram bem sucedidos e disponibilizaremos aqui algumas fotos do instalador do Xubuntu 16.04 quando carregado com o SSD travado e destravado.

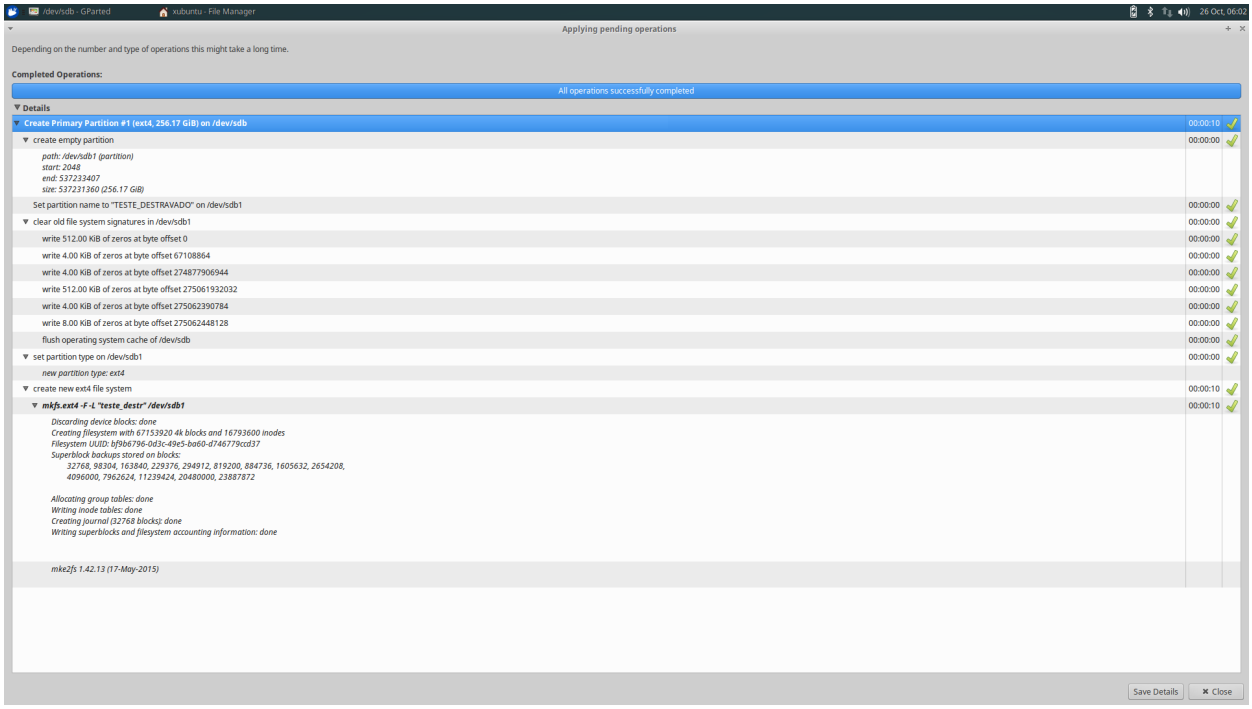
Os testes demonstram que:

1. Se o dispositivo está travado, não é permitida a escrita ou a leitura da partição escondida, apesar de ser possível saber que ela existe. Também não é possível alterar os dados na MBR que contém o PBA.

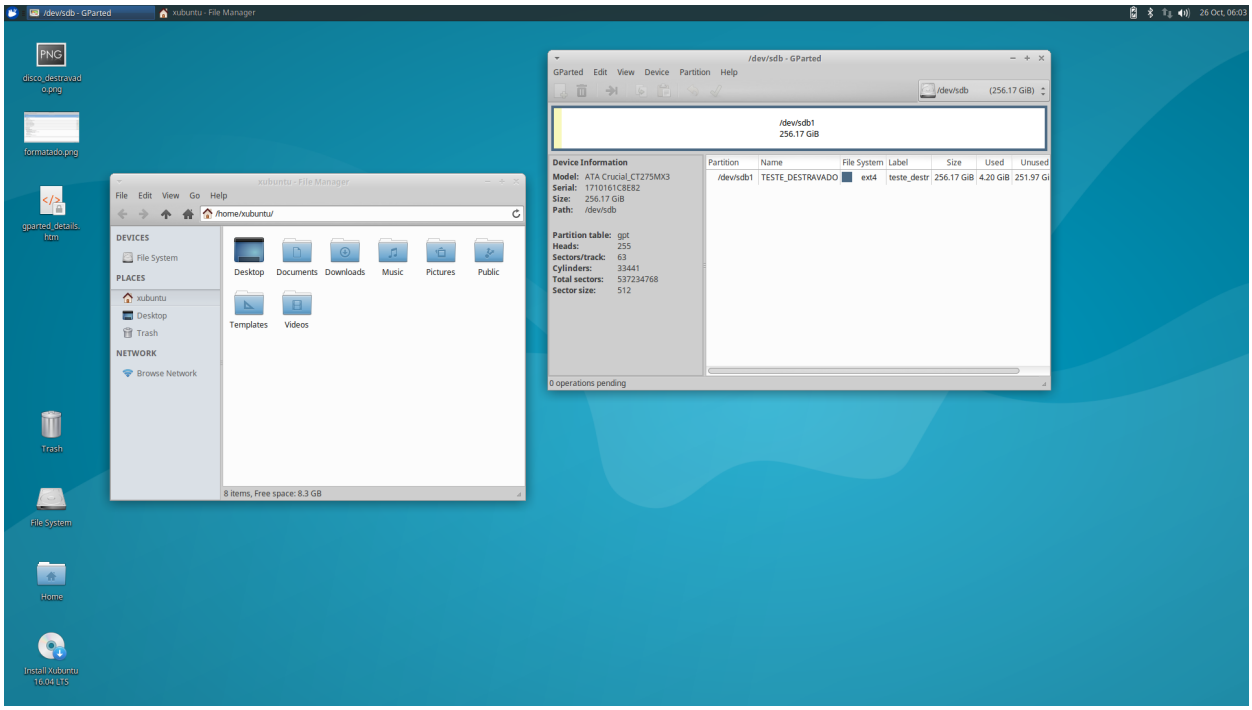
2. Se o dispositivo está destravado, é possível trabalhar normalmente no SSD.



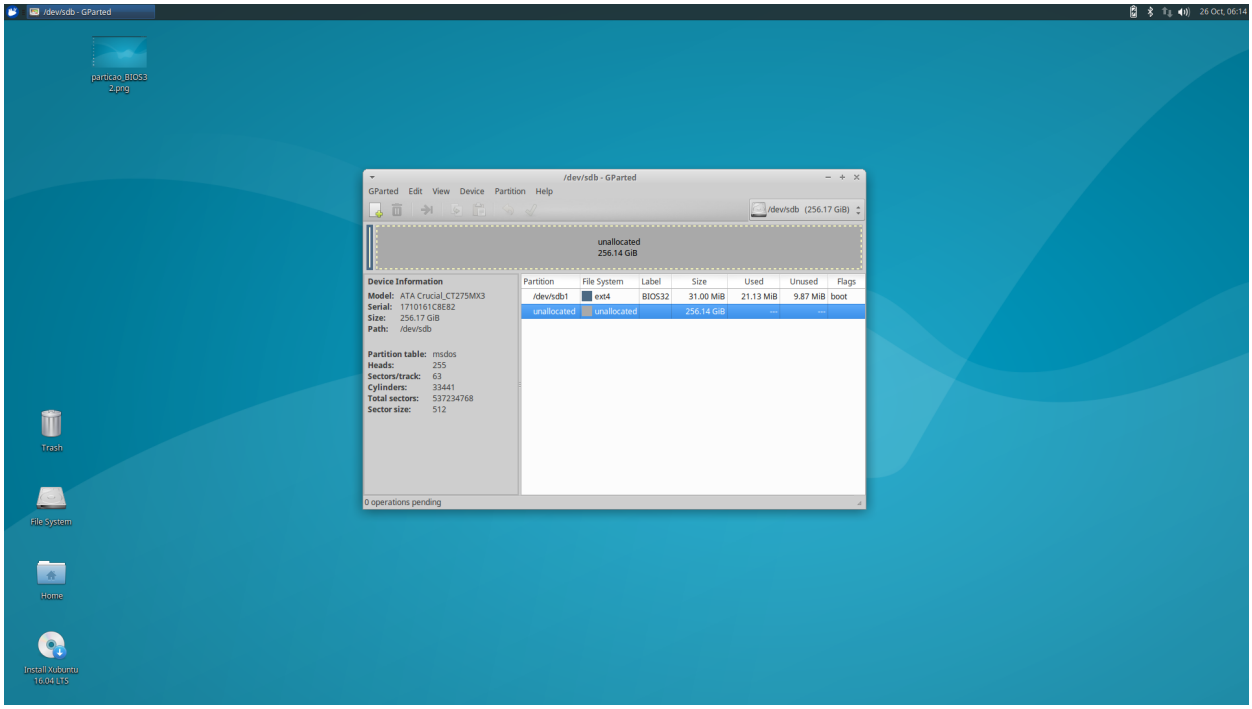
(a) *Quando o SSD está destravado, a partição que contém a MBR com PBA não é exposta.*



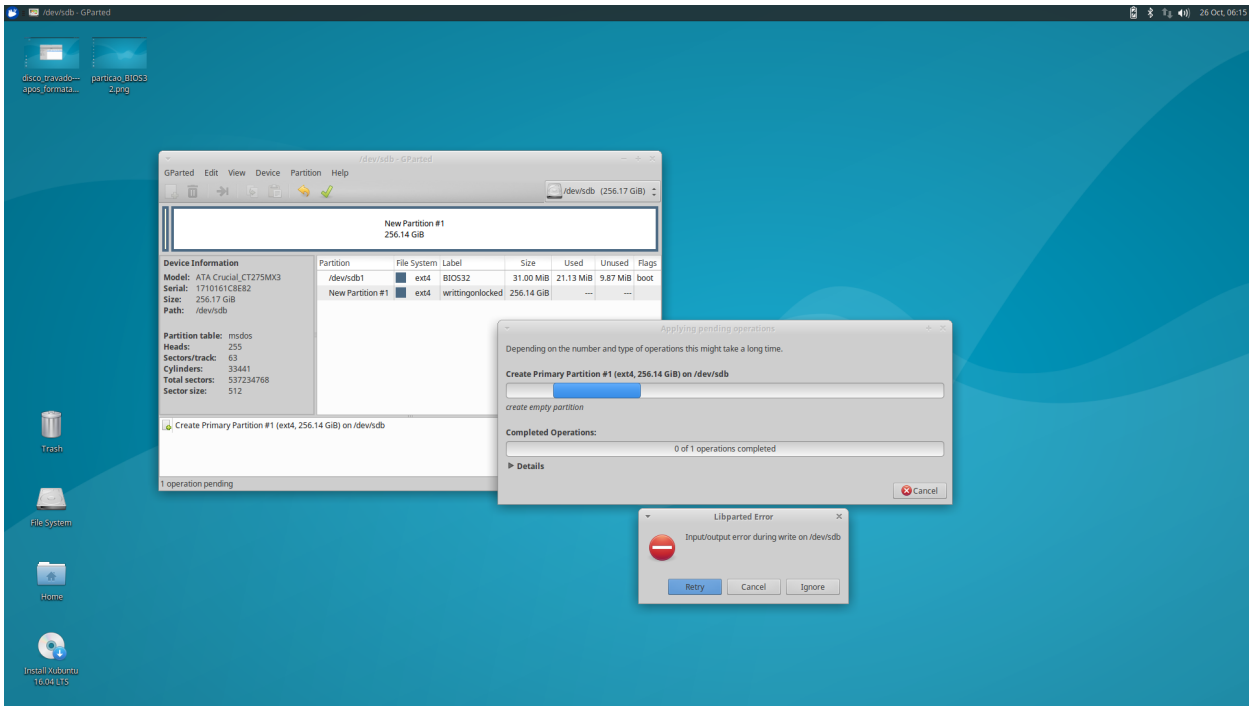
(b) *Formatação bem sucedida quando disco está destravado*



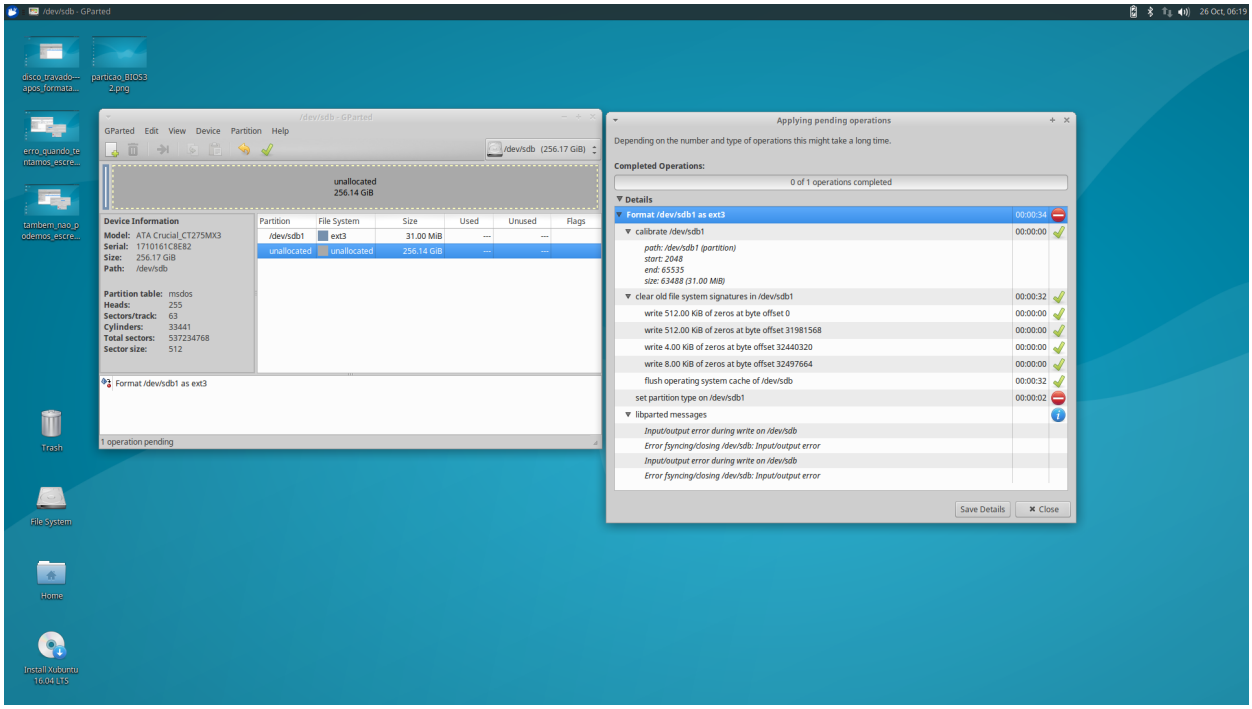
(c) *Visualização do término da formatação do SSD destravado*



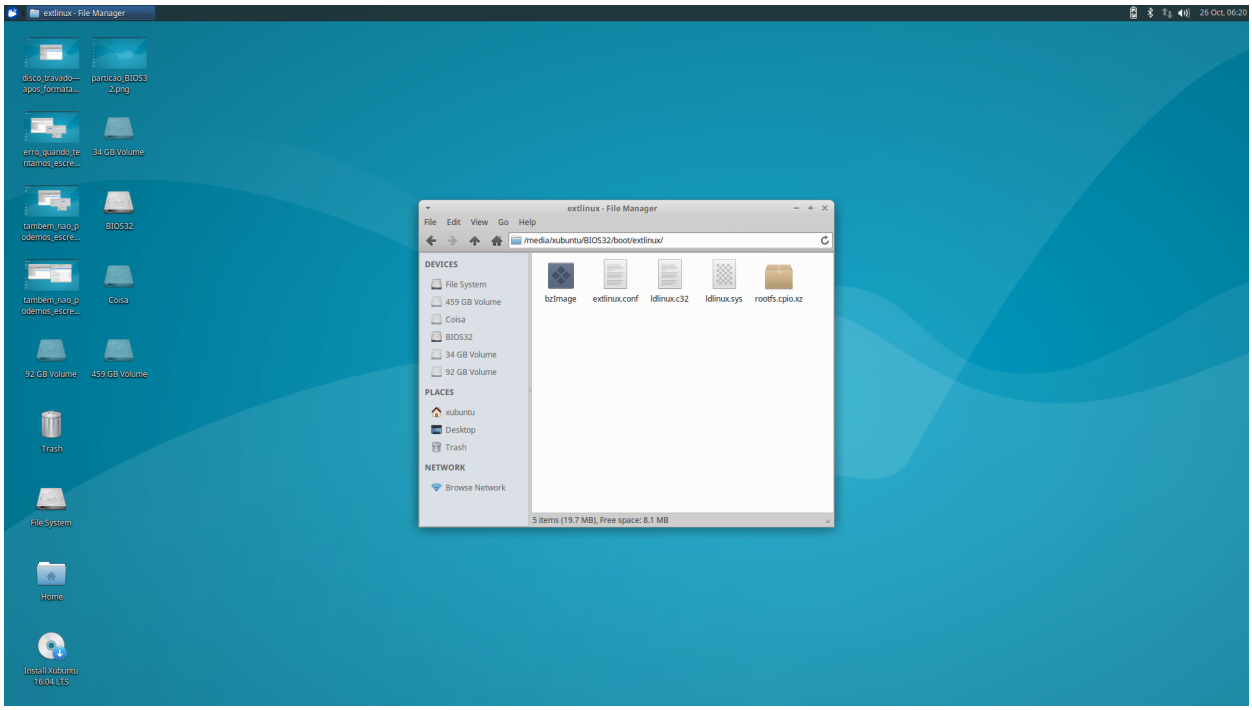
(d) *SSD travado disponibilizando conteúdo da MBR contendo PBA e partição maior porém inacessível.*



(e) *Escrever na partição travada retorna erro.*



(f) *Também não podemos escrever na partição de boot.*



(g) *Dados intactos após tentativa de formatação da partição travada de boot (que contém PBA).*

Figura 3.8: *Testes com SSD travado e destravado usando instalador do Xubuntu 16.04*

Mostramos assim que os métodos utilizados para proteção de dados em HDD e SSD estudados neste trabalho são divididos em duas classes distintas: baseada em *software* e baseada em *hardware*. Enquanto o *cryptsetup* tendo as chaves gerenciadas pelo *LUKS* se baseia em *software* pois todas as instruções são executadas no processador a pedido explícito do Sistema Operacional, o SEDutil se baseia em *hardware* por implementar a especificação dada por *TCG OPAL 2.0*, a qual o próprio dispositivo se encarrega de criptografar e decriptografar os dados quando requisitados em ambos momentos de leitura ou escrita.

Capítulo 4

Análise de Desempenho

Nesta sessão, apresentaremos os resultados obtidos a respeito das taxas de transferência de dados em dispositivos HDD e SSD usando criptografia tendo como referência as taxas obtidas quando os mesmos não estão usando criptografia. Todos os testes foram executadas em uma máquina com processador *Intel i7-6700HQ*, 16GB de RAM DDR4 2133 MHz, SSD *Samsung 850 PRO* de 512GB e HDD *Western Digital WD500BEKT* de 500GB em um *notebook Micro-Star International Co., Ltd. GE62 6QD* com *Ubuntu 16.04.3 LTS (Xenial Xerus)*.

4.1 Metodologia

Para a coleta de nossos dados, usamos a ferramenta `dd` sempre descartando *caches* de operações anteriores (*flush* total de memória) e foi passado o parâmetro `conv=fdatasync` forçando o término da escrita física do arquivo de saída antes do `dd` terminar. Para descartes de *cache* foi executado o comando:

```
sync
echo 3 > /proc/sys/vm/drop_caches
```

Tanto para escrita quanto para leitura foi utilizada uma imagem *ISO* de um DVD de instalação *Debian* cujo tamanho é 3,8GB. A escolha de um arquivo com tal tamanho se baseia na circunstância de múltiplas operações de escrita ou leitura a fim de se ter uma quantia razoável de escritas/leituras. Além disso, foi criada uma partição *ramdisk* de tamanho suficiente (10GB) para comportar arquivos temporários. Quando do ato de coleta da taxa de leitura dos dispositivos, as escritas foram todas feitas na *ramdisk*. Já para escrita, a leitura foi feita a partir da *ramdisk*. Utilizando-se do artifício de temporários em RAM, podemos descartar os atrasos que poderiam surgir por mídias auxiliares pois operações de E/S em RAM têm ordem menor em relação as outras mídias.

4.2 Resultados

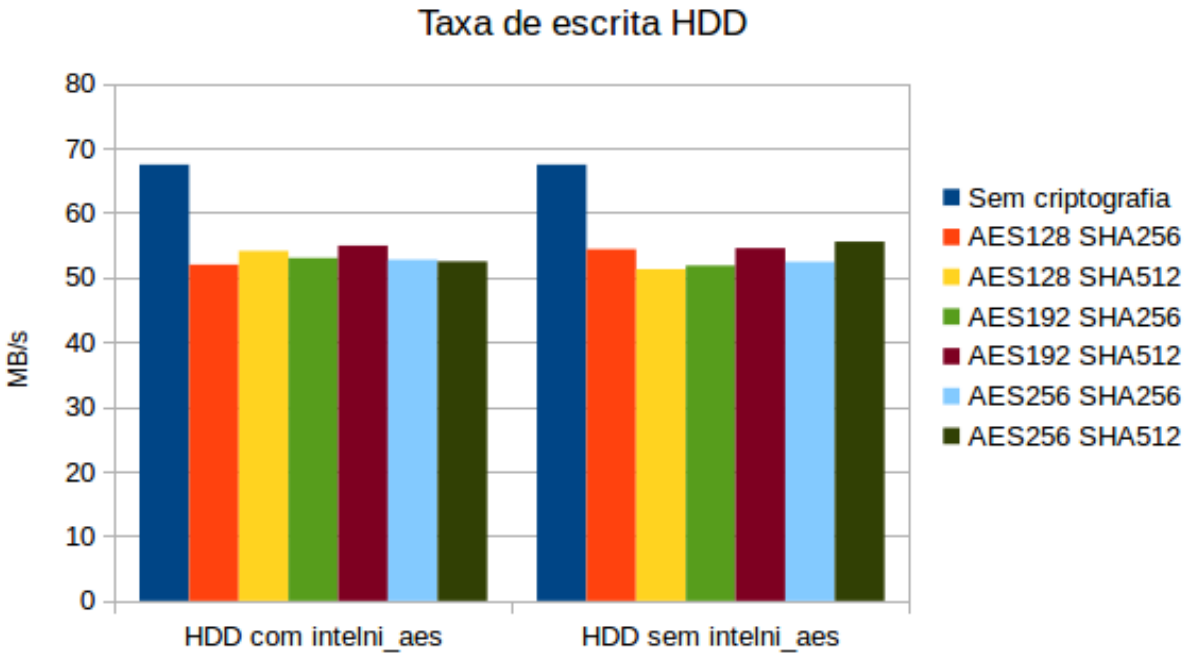
Os resultados de nossos experimentos no que diz respeito a ação de escrita podem ser vistos nas figuras 4.1a e 4.1b. Já para leitura, as figuras 4.1c e 4.1d podem ser consultadas.

Podemos observar, pelos resultados obtidos, uma queda de desempenho variante (dependendo da configuração do SHA e do AES) entre 18,2% e 22,6% com suporte nativo a AES e entre 17,28% e 23,73% sem suporte nativo a AES na escrita quando usado o *cryptsetup* no HDD testado. Já para nosso SSD temos as seguintes reduções:

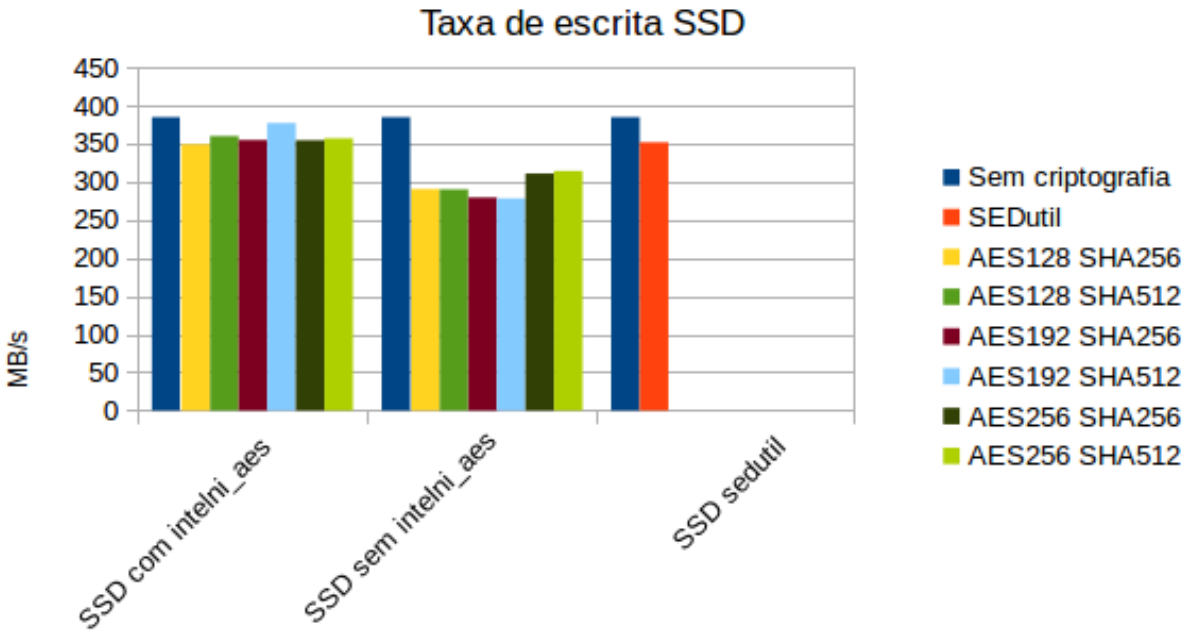
1. entre aproximadamente 2% e 8,5% quando usado o suporte nativo a AES (*cryptsetup*).
2. entre aproximadamente 18% e 27% quando não usado o suporte nativo a AES (*cryptsetup*).
3. aproximadamente 8% quando utilizada a criptografia nativa (OPAL)

Para leitura, nosso HDD teve queda variável entre aproximadamente 0% e 21% utilizando suporte nativo a AES e entre aproximadamente 7% e 21% quando desligamos o suporte nativo a AES. Já para nosso SSD temos as seguintes reduções:

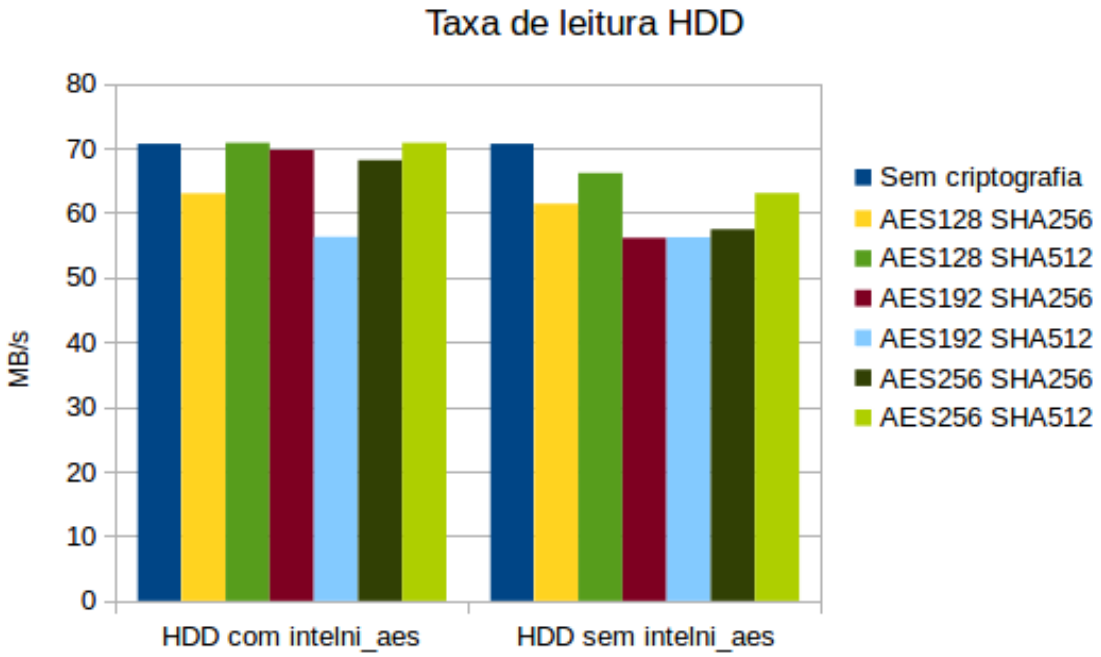
1. entre aproximadamente 4,5% e 8,7% quando usado o suporte nativo a AES (*cryptsetup*).
2. entre aproximadamente 54.9% e 60.7% quando não usado o suporte nativo a AES (*cryptsetup*).
3. menor que 0.1% quando utilizada a criptografia nativa (OPAL).



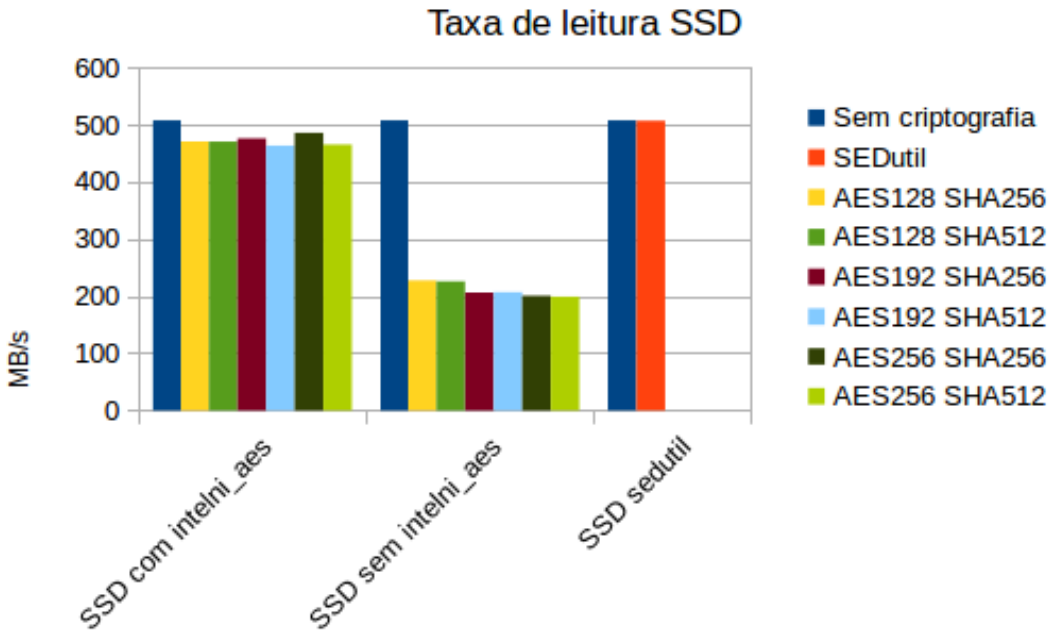
(a) *Análise de desempenho (Escrita em HDD)*



(b) *Análise de desempenho (Escrita em SSD)*



(c) *Análise de desempenho (Leitura em HDD)*



(d) *Análise de desempenho (Leitura em SSD)*

Capítulo 5

Vulnerabilidades

Neste capítulo descreveremos como técnicas de FDE estão suscetíveis a três modos de ataques comumente relatados em eventos e artigos de segurança de informação para soluções via *software*, como é o caso quando é usado *dm_ crypt*, e para soluções baseadas em *hardware* (utilização de dispositivos que seguem OPAL versão superior ou igual a 2). São eles *Ataque DMA*, *Cold Boot* e *Evil Maid*.

Primeiramente será explicado em linhas gerais o que é cada um desses ataques, qual a via de ataque, o propósito e porque cada um deles é possível num dado cenário. Julgamos necessária essa introdução pois não assumimos que o leitor seja previamente familiar com os termos apresentados nesse capítulo nem, principalmente, com as particularidades dos ataques.

Em seguida, trataremos de trazer ao leitor como cada tipo de ataque pode ser aplicado a FDE nos contextos de solução via *software* e *hardware*.

5.1 Background

5.1.1 DMA

5.1.1.1 O que é DMA

Para que seja entendido como é dado um ataque via DMA, é necessária uma introdução sobre o que significa DMA e o motivo de seu uso.

De forma simples, *Direct Memory Access* (DMA) é um mecanismo criado para transferir dados diretamente entre um dispositivo e a RAM havendo mínima participação necessária da CPU para que esta esteja disponível à outras tarefas enquanto a transferência não seja dada por completa.

A priori, temos duas técnicas (*polling* e interrupções de E/S) que possibilitam um dispositivo a se comunicar com o processador. Em conjunto, formam a base para que possamos implementar dois métodos de transferência de dados entre um dispositivo de E/S e a RAM. Porém, devido a necessidade de que haja sobrecarga por parte do processador ou do Sistema Operacional, essa abordagem se torna justificada em dispositivos de baixa capacidade de transferência (*low-bandwidth*) dado que estamos interessados em reduzir o custo dos controladores nos dispositivos. Ambas as técnicas baseadas (*polling* e interrupção de E/S) delegam a responsabilidade de mover dados e gerenciar o processo de transferência ao processador.

Usando *polling*, fazemos com que o processador tenha total domínio da transferência dos dados como por exemplo em uma aplicação de tempo real onde o processador carrega todos os dados do dispositivo e transfere para a memória.

Em contra partida, podemos arquitetar a aplicação sendo baseada em interrupções. Nessa abordagem, o Sistema Operacional transfere todos os dados em blocos menores. A cada instrução de cópia de dados, o S.O. troca de contexto e permanece gerenciando outras aplicações enquanto não haja uma interrupção do dispositivo em questão informando que mais um bloco foi copiado (com sucesso ou não). Ao final da cópia bem sucedida do último bloco, o Sistema Operacional informa a aplicação a qual requisitou a transferência que a mesma se finalizou.

Podemos ver que em ambos os casos há uma participação considerável do processador e do S.O.. Na primeira, há intenso uso do processador, uma vez que entra em um estado bloqueado esperando pelo término de toda transferência. Já na segunda, o processador tem mais liberdade para operar em outras aplicações enquanto há movimentação de dados no barramento.

Todavia, mesmo se utilizando o método de interrupção, a sobrecarga tornaria a espera impraticável se o dispositivo trabalhado é um disco rígido ou SSD devido ao volume de dados.

Dada a necessidade de haver uma maneira mais eficaz de obter a transferência de dados entre um dispositivo e a memória RAM, foi criada uma técnica que utilizasse minimamente o processador em toda a ação.

A implementação da DMA se dá com uma controladora especializada a qual gerencia toda a movimentação de dados entre os agentes e é regido pela seguinte implementação.

O processador prepara a DMA informando o identificador do dispositivo, a operação, o endereço de memória (RAM) e o volume de dados. A DMA então inicia a cópia de dados entre memória e dispositivo (se necessárias múltiplas cópias por limitação de banda, é responsabilidade da controladora da DMA orquestrar tal operação como um todo e realizar ajustes necessários no que diz respeito a endereços de memória). No momento do término de toda a transferência, a controladora da DMA gera uma interrupção e então o processador verifica o estado da transferência.

Note que um sistema pode conter mais de um dispositivo DMA em um computador. Como exemplo, podemos ter uma DMA para cada via de E/S em um sistema com uma única via de comunicação entre processador e a RAM e múltiplas vias de entrada e saída.

Para mais informações sobre DMA, sugerimos a leitura de Patterson (2011)

5.1.1.2 Ataque DMA

Como descrito em 5.1.1.1, DMA é uma tecnologia usada para transferência de dados entre um dispositivo e a RAM sem grandes intervenções do processador. Porém, o modo como essa tecnologia foi arquitetada traz a possibilidade de leitura e escrita diretamente na RAM sem restrição alguma (Dornseif *et al.* (2004) e Dornseif *et al.* (2005)) oferecendo vazamento de informação como roubo de chaves criptográficas, senhas, monitoramento de processos em execução na máquina alvo ou até mesmo desbloqueio de telas cujo propósito é a proteção enquanto a máquina não está sendo vigiada por alguns instantes https://github.com/carmaa/inception/, instalação ou execução de *spyware*, *backdoor* entre outros.

As vias mais exploradas nesse tipo de ataque são as portas *FireWire*, *Thunderbolt* e PCI ou PCI-e. Se um sistema estiver totalmente desprotegido contra esse tipo de ataque, este é dado de forma simples. Basta inserirmos algum dispositivo malicioso em uma porta que suporta DMA e solicitar a leitura ou escrita da RAM. Caso um tipo mais elaborado de injeção de código seja desejado, basta mapearmos o local alvo da memória, escalar privilégio para escrita e inserir dados alterados como Böck (2009) demonstrou em sua prova de conceito (PoC - *Proof of Concept*). Um dos ataques para desbloqueio da tela de proteção se deu

simplesmente sobrescrevendo o código que verifica a senha por instruções NOPs. [1]

Boileau (2006) relata que máquinas do tipo caixa eletrônicos também podem ser alvos desse tipo de ataque, o que sustenta o argumento da sua seriedade.

Ressaltamos também que, para explorar essa falha, a máquina alvo deve se encontrar ligada e com o S.O. em execução. Caso contrário não podemos instruir o processador a nos fornecer informações da RAM.

Atualmente, máquinas mais recentes dispõem de um mecanismo chamado IOMMU 6.1.2 que pode ser usado para prevenir esse tipo de ataque, a IOMMU não protege totalmente um sistema por dois motivos. A saber: 1) IOMMU provê segurança somente nas páginas mapeadas deixando *buffers* que residem na mesma página desprotegidos; 2) devida a demora com a qual as páginas são removidas do mapeamento, uma janela é aberta na qual dispositivos podem acessar conteúdos em memória como demostrado por Markuze *et al.* (2016).

5.1.2 Evil Maid

De forma simples, a inicialização (*boot*) de uma máquina se dá na seguinte ordem: a BIOS realiza verificações de sanidade nos dispositivos como processador e módulos de memória, verifica qual dispositivo deve ser carregado para inicialização, evoca a MBR do dispositivo (a qual tem informações como números de partições que há no mesmo, tamanho, localização, formatação etc. das mesmas), o dispositivo devolve quais partições são inicializáveis (*bootable*) e a inicialização é dada em uma delas. Então é iniciado o processo de carregamento do Sistema Operacional para RAM através de um *boot loader* como o *GRand Unified Bootloader* (GRUB) ou *Windows's BOOTMGR*.

Portanto, a MBR deve dispor seu conteúdo de forma não criptografada no momento de *boot* para que a leitura seja dada. E, precisamente por conta dessa premissa, que o *Evil Maid* se dá com sucesso.

Em 2009 Rutkowska descreve como foi possível atacar o *TrueCrypt* [2] analisando a MBR e aplicando modificações no binário de leitura de senha caso houvesse uma assinatura válida de cabeçalho *TrueCrypt* Rutkowska (2009).

Rutkowska detalha: suponha que há um computador com um disco criptografado pela forma *TrueCrypt* desligado, tornando assim o *Cold Boot*5.1.3 impraticável. Nessas condições, é possível reescrever a MBR a fim de inserir um *keylogger* [3] que seja executado no momento exato da leitura da senha quando dá-se a inicialização do sistema. Essa inserção de código malicioso pode ser feita de dois modos e depende do fato de o sistema a ser atacado solicitar ou não senha para inicializar a BIOS.

Se o sistema não solicitar senha para inicializar a BIOS, podemos trocar a fonte de inicialização do sistema para um *pendrive* que contenha um programa capaz de ler a MBR, descompactar as informações ali presente, verificar a presença do cabeçalho *TrueCrypt*, executar busca pela função que faz a leitura da senha e modificá-la para que, além de retransmiti-la para o carregar do *TrueCrypt*, seja armazenada em um local específico da MBR (note que a senha deve ser armazenada na MBR pois permanece decriptografada), recalcule o tamanho e *checksum* do cabeçalho para que seja reescrito (permanecendo assim válido).

[1] NOP, *no operation*, é uma instrução em assembly que "não desempenha operação". Na arquitetura Intel, "é uma instrução de um-byte ou outra de múltiplos bytes que consome espaço no fluxo de instruções mas que não afeta o contexto da máquina, exceto o conteúdo do registrador EIP."Mais informações em Intel (2016) pg 1289

[2] **TrueCrypt** foi um projeto cuja proposta era criar um disco virtual e criptografado através de um arquivo usual e criptografar uma partição inteira ou o disco inteiro http://truecrypt.sourceforge.net/.

[3] programa o qual armazena todas as entradas do teclado

Caso o sistema solicite senha, não podemos simplesmente remover a bateria da placa mãe pois causaria suspeita, mas podemos remover a unidade do computador alvo, inserir em um segundo, aplicar o código e reinserir no alvo.

Em sua descrição de como atacar o *TrueCrypt*, Rutkowska afirma que no primeiro caso, o ataque consome cerca de dois minutos, tempo necessário para que uma pessoa disfarçada de camareira possa entrar no quarto de hotel onde a vítima está hospedada e executar o ataque. Em sua implementação mais simples, esse tipo de ataque requer duas visitas ao alvo - uma para executar a modificação da MBR e outra para leitura da senha. Porém é possível adaptar o ataque para que ocorra envio da senha por rede em tempo real dado que ocorreu uma duplicação do volume no momento da primeira visita.

5.1.3 O que é Cold Boot

O método *Cold Boot*, também por vezes conhecido como *plataform reset attack*, *cold ghosting attack* ou *iceman attack* se baseia no fato de que informações em memória RAM, dita como voláteis, não se perdem no exato momento em que a máquina é desligada.

Hoje, é sabido que o decaimento das informações (*bits*) contidos na RAM se dá de forma contínua ao longo do tempo até atingir um estado de perda total da informação. Chamamos esse fenômeno de *remanência de dados* e tal hipótese foi levantada no final da década de 70 e início dos anos 80 por Link e May em 79 (Gruhn e Müller (2013) e Jaeger e Blalock (1990)).

Um dos primeiros estudos sobre remanência de informação em *Static Random-Access Memory* (SRAM) e *Dynamic Random-Access Memory* (DRAM) de forma mais categórica deu-se por Gutmann (1996). Retirando a alimentação da memória e levando sua temperatura a menos de -60°C, informações poderiam ser extraídas mesmo após semanas. Em 2001, Gutmann dá uma atenção maior para esse fato e demonstra que é possível ler informações da RAM em momentos posteriores a retirada de alimentação e alerta que variáveis criptográficas devam permanecer em memória o menor tempo necessário para que os componentes eletrônicos tenham a menor "memória"possível sobre os dados em uso e, enquanto em uso, os bits devem ser invertidos de tempos em tempos para evitar a leitura posterior dos dados Gutmann (2001).

Apesar de não ter sido demonstrado por Gutmann et. al, no final dos anos 2000, utilizando-se do mesmo princípio de refrigeração, Halderman *et al.* (2008) descreve como foi possível atacar FDEs incluindo *BitLocker*, *TrueCrypt* e *FileVault* se a máquina estivesse no modo suspenso ou com a tela bloqueada com senha de usuário em sessão corrente, *logged in*. Também foi mostrado que é possível extrair chave privada RSA de servidores *Apache*.

Ao longo dos anos, com a evolução das tecnologias utilizadas, os componentes eletrônicos vêm tendo tempo de remanência cada vez menor. Gutmann em 2001 pôde experimentar decaimento que perdurava por tempo demasiado longo. Já Haldermann et. al obteve um decaimento de 1 por cento dos bits da memória após 10 minutos caso fosse atingida uma temperatura de -50°C e de 0,17 por cento a -196°C.

Um dos estudos mais recentes nessa área foi dado por Yitbarek *et al.* (2017). Em seus resultados, Yitbarek et. al mostram que mesmo nos dias de hoje é possível executar com sucesso ataques *Cold Boot* em máquinas que utilizam-se de memórias DDR4 [4], com ou sem *Dual Channel* e com ou sem embaralhamento de dados na RAM, funcionalidade provida diretamente pelo processador [5]

[4] tipo mais recente no momento para memória RAM de uso comercial

[5] Esse embaralhamento não tem propósito de segurança de informação mas sim de tornar a transmissão mais eficaz Yitbarek *et al.* (2017)

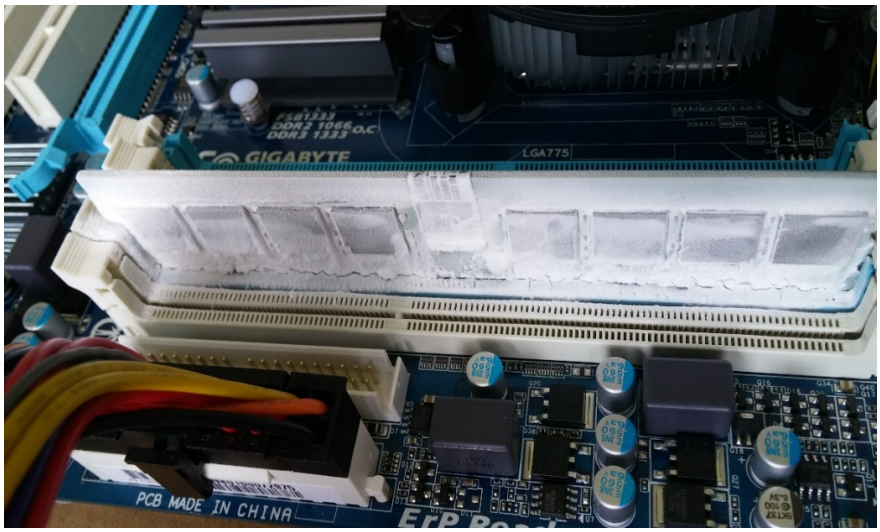


Figura 5.1: *Módulo de memória SDRAM congelada para leitura*

Entretanto, apesar de promissor, o método *Cold Boot* deve ser aplicado com cautela e como último recurso devido a sua facilidade em perder os dados e dificuldade de aplicação atrelada às variáveis correlacionadas. Por exemplo, se um investigador tentar medir a temperatura ou capacitância da placa mãe de um sistema, seja por introdução de instrumentos no computador ou pela leitura do estado da BIOS, muito provavelmente ele será mudado Carbone (2011).

A figura 5.1[6] ilustra um módulo de memória retirado da máquina alvo e inserido em outra para leitura.

5.2 Atacando FDEs

5.2.1 Em cryptsetup com LUKS

Como esse método se baseia em software, as chaves devem permanecer em RAM para que ações criptográficas sejam executadas a todo momento pelo processador, uma vez que não há suporte por parte do dispositivo em questão. Sendo assim, não há necessidade de alterações nos métodos conhecidos e descritos em 5.1.1.2, 5.1.2 e 5.1.3.

Caso o alvo esteja usando LUKS, dirigimos o ataque conforme o estado da máquina.

1. Ligada ou Suspensa com RAM alimentada (*ACPI G0, S1, S2, S3* [7]). Nessas circunstâncias, podemos primeiramente utilizar ataque a DMA ou *Cold boot* dado insucesso do anterior.

2. Para os outros estados, podemos utilizar o método *Evil Maid*.

5.2.2 Em OPAL

Descritos os ataques nas sessões anteriores, continuamos os estudos agora descrevendo como podemos atacar dispositivos protegidos por OPAL.

[6] Retirada de Lindenlauf *et al.* (2015)

[7] *Advanced Configuration and Power Interface* http://www.acpi.info/DOWNLOADS/ACPI_5_Errata%20A.pdf páginas 25-28

Uma vez que as chaves criptográficas não permanecem em RAM, ataques do tipo *Cold Boot* não se aplicam nesse contexto. O máximo de informações que poderíamos obter lendo a RAM congelada seria uma foto de processos rodando, nome de arquivos abertos, caminho de *sockets* com informações de rede quando a máquina estava ligada e eventuais conteúdos de arquivos que estavam abertos para leitura ou escrita. Mas não o acesso total ao SSD.

Porém alguns tipos de ataques ainda se fazem vitoriosos em SSD com OPAL 2.0.

Evil Maid

Como dito, podemos evoluir o ataque *Evil Maid* para uma única visita. Caso o atacante consiga substituir o SSD da máquina alvo com um modelo similar o ataque segue da seguinte forma:

1. Habilite OPAL e a trava no SSD similar.
2. Modifique o PBA para não abrir o SSD similar e:
   (a) Conectar-se a uma rede pré-estabelecida (*hot spot* do celular por exemplo).
   (b) Enviar a senha para um IP estabelecido.
   (c) Caso sucesso, auto modificar-se para remover rastros. Caso contrário, mantenha seu comportamento.
3. Escreva esse PBA na MBR do SSD similar.
4. Insira o SSD falso na máquina alvo.

Hot Plug

Esse método de ataque faz um paralelo ao *Cold Boot*. Nele, podemos explorar uma falha exposta por parte dos fabricantes de dispositivos. Müller *et al.* (2012) discorrem sobre tal ataque em servidores e *laptops*.

De posse da máquina ligada, podemos, em algumas máquinas, retirar somente o cabo SATA (de dados) do SSD, ligar em outra máquina e ler de forma trivial todos os dados pois, quando retiramos somente o cabo SATA, o SSD não teve perda de energia fazendo com que, assim, não entrasse no modo travado.

"Como consequência, se um sistema estiver em execução, SEDs são geralmente mais *inseguros* que FDE baseados em *software* devido a facilidade e efetividade de ataques *hot plug*. Nessa perspectiva, um SED ligado se comporta mais como um '*self-decrypting device*' do que como um '*self-encrypting device*'- Müller *et al.* (2012).

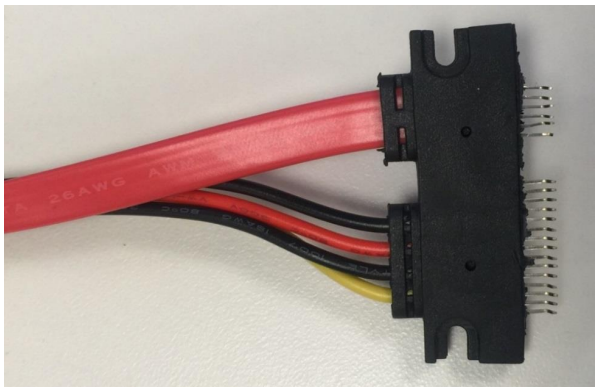
O uso de *Hot plug* é mais fácil em sistemas *desktop* / servidores devido a extensão e flexibilidade dos cabos que serão trabalhados. Acessos a esses cabos também são mais favoráveis devido à própria construção física da máquina.

Porém, em *laptops*, o ataque sofre um ganho de dificuldade. Alguns dos motivos são **a)** Não haver cabos ligando o dispositivo à máquina pois ele é ligado diretamente à placa

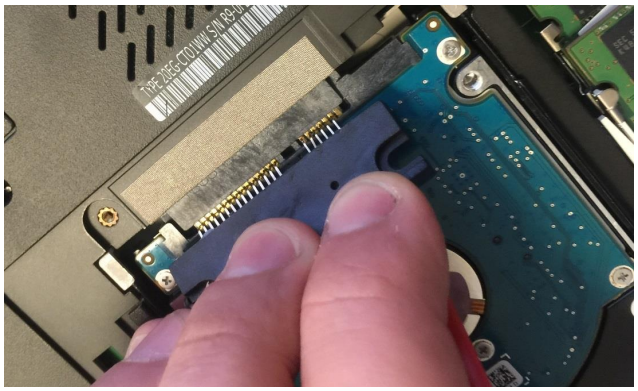
como visto na figura 5.2a **b)** Não estão ligados o tempo todo (24/7) como servidores; geralmente estão desligados ou suspenso, travando assim o SSD[8].

Em *laptops*, então, se faz necessário uso de cabo SATA adaptado como na figura 5.2b

(a) *Dispositivo SATA ligado diretamente à placa mãe.*



(b) *Cabo SATA adaptado para ataque hot plug*



(c) *Cabo adaptado e inserido na máquina alvo*

Figura 5.2: *Cabo adaptado para hot plug. Imagens retiradas de Boteanu e Fowler (2015)*

DMA

O ataque DMA em SEDs se dá de forma análoga a FDE por *software*. Caso a máquina alvo tenha alguma porta que suporte DMA, pode ser possível o uso da ferramenta *inception*[9]. Após o destravamento da tela, a leitura de todos os dados se mostra trivial. Um pré requisito para esse ataque é que a máquina esteja ligada com o dispositivo destravado.

Cold Boot

Como as chaves criptográficas em SED não permanecem em RAM, é inútil tentar rei-

[8] Apesar de algumas BIOS armazenarem a senha do dispositivo nela mesma possibilitando a inserção de cabos após forçar o *laptop* entrar em suspensão, essa abordagem não se aplica em proteção usando OPAL no SEDutil, somente para segurança SATA comuns. OPAL está seguro nesse caso.

[9] Ferramenta usada para destravar a tela que protege o sistema enquanto ele está ligado.https://github.com/carmaa/inception/

niciar a máquina carregando um Sistema Operacional modificado para leitura da RAM. Porém muitas placas-mãe não possibilitam pedido de senha para reiniciar a máquina e não interrompem ligeiramente a energia do SSD fazendo com que possamos carregar outro SO com o SSD destravado e ler toda informação. Caso a senha seja pedida, essa abordagem se encontra comprometida pois não podemos desligar a máquina, retirar a bateria da placa-mãe para que não haja senha e ligarmos novamente.

Baseado na árvore de decisão para ataques a SEDs de Müller *et al.* (2012) (que não aborda OPAL), a figura 5.3 apresenta uma nova árvore de decisão para ataque SED com OPAL.

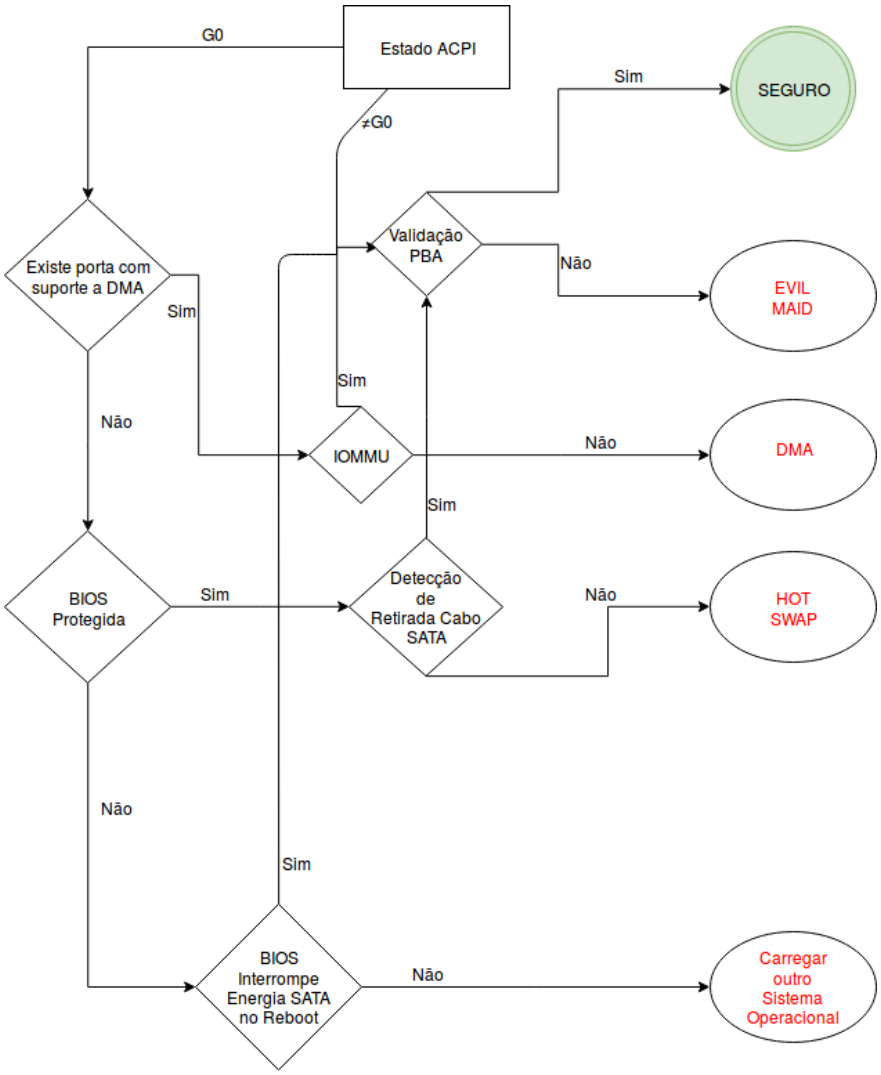


Figura 5.3: *Árvore de decisão para ataque a OPAL*

Capítulo 6

Prevenções

6.1 Background

6.1.1 VT-x/AMD-V

Intel *Virtual-Machine Extensions* (VMX), ou popularmente VT-x, é um conjunto de instruções adicionado aos processadores Intel para dar melhor suporte à virtualização de máquinas fazendo com que haja maior isolamento entre as Máquinas Virtuais (MVs) e que elas tenham a ilusão de ter total domínio sobre a máquina (estar sendo executada com privilégio de nível 0 [1]) as novas instruções são: INVEPT, INVVPID, VMCALL, VMCLEAR, VMFUNC, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON e suas descrições podem sem encontradas em Intel (2016). De forma análoga dá-se a implementação para AMD-V.

6.1.2 IOMMU/VT-d

Input/Output Memory Management Unit (IOMMU) em processadores AMD, *Virtual Technology for Directed I/O* (VT-d) em processadores Intel, é uma tecnologia criada para dar melhor suporte para máquinas virtuais fazendo com que dentre muitas melhorias, propõe-se a (AMD (2016) e Intel (2016)):

1. Permitir que máquinas virtuais acessem diretamente dispositivos de entrada e saída.
2. Prover melhor proteção ao Sistema Operacional *Host* restringindo dados incorretos ou não intencionais por parte de *drivers* mal programados sejam levados aos dispositivos, permitindo maior robustez ao S.O..
3. Filtrar e remapear interrupções.
4. Compartilhar um espaço de endereço virtual de um processo com um periférico selecionado.
5. Isolamento da DMA: Sistema Operacional pode gerenciar tarefas de Entrada e Saída criando um mapeamento onde, quando requisitado por um dispositivo de E/S, em vez de acessar diretamente a RAM, o S.O. repassa o mapeamento virtual designado ao dispositivo, tentando assim eliminar ataques a DMA.

[1] Para mais informações sobre níveis de proteção, consultar Karger e Herbert (1984) ou https://en.wikipedia.org/wiki/Protection_ring

6.1.3 BitVisor

BitVisor é um *hypervisor* desenhado com o propósito de ter um código mínimo que permita que grande parte das requisições de E/S requisitadas pelo S.O. residente na M.V. sejam retransmitidas diretamente ao dispositivo em questão com a menor virtualização possível de dispositivos. Funcionalidades de segurança também são completamente asseguradas pelo *hypervisor* num conjunto mínimo. Tal arquitetura usa *drivers* de dispositivos no S.O. da M.V. para gerenciar os dispositivos físicos, assim reduzindo o conjunto de componentes virtuais que o *hypervisor* deve disponibilizar. Tal arquitetura suporta somente uma única M.V., eliminando assim a necessidade de compartilhamento e proteção quando há compartilhamento. BitVisor é projeto de Shinagawa *et al.* (2009) https://www.bitvisor.org/.

6.2 TreVisor

Nesta sessão, discorreremos sobre dois métodos propostos para prevenção de ataque a DMA. Apesar de aparentar uma solução para somente FDE baseado em *software*, é importante sua adoção em SED pois torna o sistema como um todo mais protegido.

Dado todo contexto descrito nas sessões anteriores sobre como se dá um ataque a DMA e visto que ainda hoje tal exploração é praticável, um método para contornar roubo de chaves criptográficas de FDE baseado em *software* é migrar a chave da RAM para registradores da CPU. Com essa atitude, leituras não autorizadas na RAM, mesmo sendo algo indesejado, não põe o disco inteiro em risco. Tal estudo foi proposto e implementado por Müller *et al.* (2011) no projeto *TRESOR*[2]. Em sua publicação, Müller et. al propõem que as chaves criptográficas sejam armazenadas em registradores de depuração, assim nunca expostas a contextos externos à CPU, nem mesmo a *cache L1* (nível mais interno e mais próximo à CPU). Porém essa solução ainda não nos dá proteção contra ataque a DMA pois tal ataque geralmente permite o comprometimento do sistema como um todo e consequentemente leituras dos registradores executando códigos com a mais alta permissão, *ring 0*.

A solução para esse problema é dada utilizando VT-x (AMD-V), IOMMU (VT-d), Intel AES NI, *TRESOR* e BitVisor conjuntamente. Unidas todas essas novas ferramentas, Müller *et al.* (2012) criaram o TreVisor (TRESOR + BitVisor). TreVisor é uma solução baseada em *software* e *hardware* para FDE com o propósito de ser:

1. praticamente segura contra ameaças do tipo *Cold Boot* e DMA,
2. seguro de forma criptográfica usando criptografia AES-(128,192,256),
3. rápida, em particular quando usada com Intel AES NI.

Outra vantagem do TreVisor é o fato de podermos escolher outro S.O. e não somente GNU/Linux pois o S.O. será executado sobre a camada do *hypervisor* BitVisor. Como discutido em Müller *et al.* (2012), TreVisor trás além das vantagens acima:

1. O *hypervisor* é isolado, assim mesmo alguém com privilégio ao *ring 0* não tem acesso aos registradores que comportam as chaves criptográficas.
2. Todos Sistemas Operacionais são igualmente suportados; é possível ter acesso ao mesmo dispositivo criptografado tanto pelo GNU/Linux quanto pelo Windows.

[2]Acrônimo recursivo para *TRESOR Runs Encryption Securely Outside RAM*, pronunciado *trɛ:zoa* e também pode significar cofre em alemão

3. *Hypervisors* são pequenos tornando o risco de sérios erros de programação menores devido ao tamanho do BitVisor.

4. Construir um disco *realmente* criptografado por inteiro é simples, incluindo a MBR; somente o *hypervisor* não deve ser criptografado.

5. Ameaças contra DMA estão centralizadas e são de responsabilidade das configurações da IOMMU (VT-d).

Algumas notas sobre TreVisor no momento da publicação do trabalho dos autores do projeto:

1. Chaves criptográficas são armazenadas nos registradores de depuração *dr0* - *dr3*[3], justificado pelo fato que usuários comuns não usam esses registradores e, caso necessário, é possível ter *soft break points* para depuração.

2. Para habilitar o modo ACPI *S3*, a senha deve ser informada novamente, uma vez que as chaves nunca são armazenada na RAM. Porém, a mesma deve ser digitada às cegas devido a um problema com dispositivos de vídeo. Segundo eles, "suporte a S3 no Trevisor ainda é um trabalho em progresso".

3. Foram testadas aplicações de propósitos gerais como de escritório, de internet e jogos 3D. Todos tiveram desempenho adequado.

4. Depuradores não foram executados com sucesso; como por exemplo *GNU Debugger* (GDB) e *OllyDBG*, pois não foi possível adquirir *hardware breakpoints* pelo fato do TreVisor não permitir a sobrescrita dos registradores de depuração (comportamento esperado).

5. Virtualização é suportada mas tem um desempenho baixo pois o *hypervisor* já está se utilizando das extensões. Também segundo os autores, futuramente essa barreira poderá ser suportada pois VT-x podem ser aninhadas se o *hypervisor* mais próximo da CPU der suporte.

6. TreVisor pôde suportar Sistemas Operacionais como Windows 7, GNU/Linux, variações de BSD e outros.

7. Penalidade de desempenho no sistema pode chegar a 1/3.

Tendo as chaves fora da RAM, é fácil perceber que ataques do tipo *Cold Boot* não se aplicam. Para ataques a DMA segue a seguinte análise. As chaves se encontram em *dr0* - *dr3*, caso o S.O. virtualizado seja comprometido, mesmo em *ring 0*, esses registradores estão protegidos em uma nova camada de permissão *ring -1*, assim chamada por Müller et. al, provida pelo *hypervisor*, e cabe ao BitVisor interceptar qualquer tentativa de leitura ou escrita nesses registradores.

Com essa nova abordagem, o disco pode ser criptografado de forma *íntegra*. Mesmo a MBR pode ser criptografada pois o *bootloader* é carregado a partir de um sistema apartado da máquina num dispositivo USB por exemplo.

Esse último fato leva a defesa que o TreVisor é também protegido contra *Evil Maid*.

Porém alguns pontos devem ser levantados: **a)** TreVisor não está seguro contra BIOS modificada (*BIOS kit*[4] Müller *et al.* (2012)). **b)** As instruções de IOMMU não provêm total

[3] 4 registradores de 64 bits cada um, suportando chaves de até 256 bits

[4] Ataque que injeta código malicioso na BIOS Sacco e Ortega (2009)

segurança contra ataques a DMA pois o mapeamento é dado por páginas fazendo com que informações na mesma possam ser extraídas. Dá-se então a necessidade de uma segurança no que tange aos *bytes* dessas páginas. Tal correção pode ser atingida implementando *shadow buffers* Markuze *et al.* (2016).

Na figura 6.1, podemos ver o local de atuação das extensões de instrução da CPU e como elas protegem FDE contra os ataques do tipo DMA e Cold Boot.

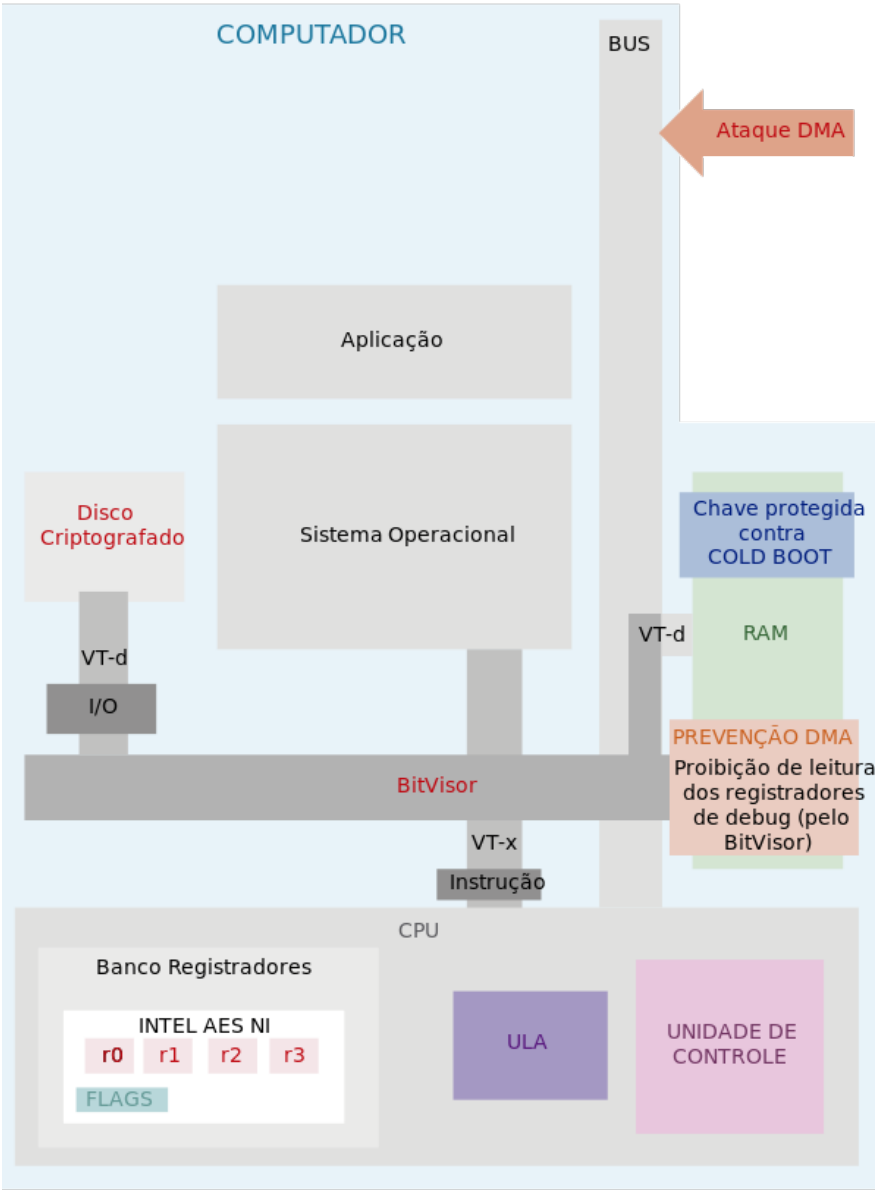


Figura 6.1: *Atuação de módulos de CPU no BitVisor*

6.3 Qubes-OS

O QubesOS (a) é um sistema operacional razoavelmente seguro[5]. Ele oferece o que chama de *segurança por compartimentalização*. Sua implementação é feita utilizando o *Xen Hypervisor* com várias máquinas virtuais independentes rodando sobre ele. Assim, se uma delas

[5] "Qubes OS: A reasonably secure operating system"

for atacada, não terá a capacidade de acessar as outras máquinas virtuais rodando no mesmo sistema. A cada um desses compartimentos é dado o nome de *Qube*.

Por exemplo, podemos ter um *Qube* para acessar sites não-confiáveis, um outro *Qube* para coisas relacionadas ao trabalho e um terceiro *Qube* para transações bancárias pela internet.

O *Qubes-OS* utiliza FDE por padrão e cria um compartimento que tem acesso ao disco. O *Qubes-OS* trabalha com a hipótese de que esse compartimento é inseguro, ou seja, se ele for comprometido, os outros *Qubes* continuam seguros. Para que isso aconteça, cada um dos *Qubes* tem uma chave criptográfica própria de acesso ao disco, disponível apenas à ele próprio e a um compartimento especial, chamado de *Dom0* (Domínio 0).

O *Dom0* é o único *Qube* cujo comprometimento afeta todo o sistema. Esse domínio é responsável pela interação com o usuário (mouse, teclado e interface gráfica). Qualquer controle desse domínio obtido por um atacante compromete completamente o sistema pois é impossível diferenciar um comando dado por um atacante de um comando dado pelo usuário a partir de seus dispositivos de entrada. Além disso, esse domínio não tem nenhum acesso ao mundo externo como, por exemplo, acesso à rede.

O *Dom0* é também responsável pela administração do *hypervisor*, ou seja, nele é executado o *daemon XenStore*.

6.3.1 Proteção contra ataque DMA

O *Qubes-OS* apresenta a proteção contra DMA intrínseca do *hypervisor*, que suporta a tecnologia VT-d. Assim, um *Qube* não tem acesso direto à memória de um outro *Qube* qualquer.

6.3.2 Proteção contra ataque Evil-Maid

Em um post no seu blog, Rutkowska (2011) descreve uma solução ao ataque *Evil Maid*. O *Anti Evil Maid*, em suas próprias palavras, consiste em "autenticar a máquina para o usuário". Isso é feito utilizando a operação *unseal* de um TPM, que garante que o *software* carregado não foi alterado. Assim, após uma operação *unseal* executada corretamente, uma certa mensagem é exibida na tela. Geralmente, utiliza-se uma imagem ao invés de um texto devido à dificuldade de se copiar e escrever *hardcoded* tal informação em um código malicioso a ser utilizado em um ataque *Evil Maid*. O *Qubes-OS* oferece a possibilidade de instalação dessa implementação do *Anti Evil Maid* (QubesOS (b)).

6.3.3 Proteção contra ataque Cold Boot

Uma das proteções contra um ataque Cold Boot é a aleatorização ou limpeza da RAM durante o desligamento do sistema ou, no caso do *Qubes-OS*, a cada *Qube* descartado. Tal funcionalidade já existe em outros sistemas operacionais com foco em segurança como, por exemplo, o Tails mas ainda está em discussão no *Qubes-OS* em uma das suas *issues* no *Github* (QubesOS (c)).

A figura 6.2 mostra como se dá o conceito de isolamento dentro do *Qubes-Os* enquanto a figura 6.3 ilustra todo o *data flow* nele[6].

[6] Imagens retiradas site do projeto https://www.qubes-os.org/

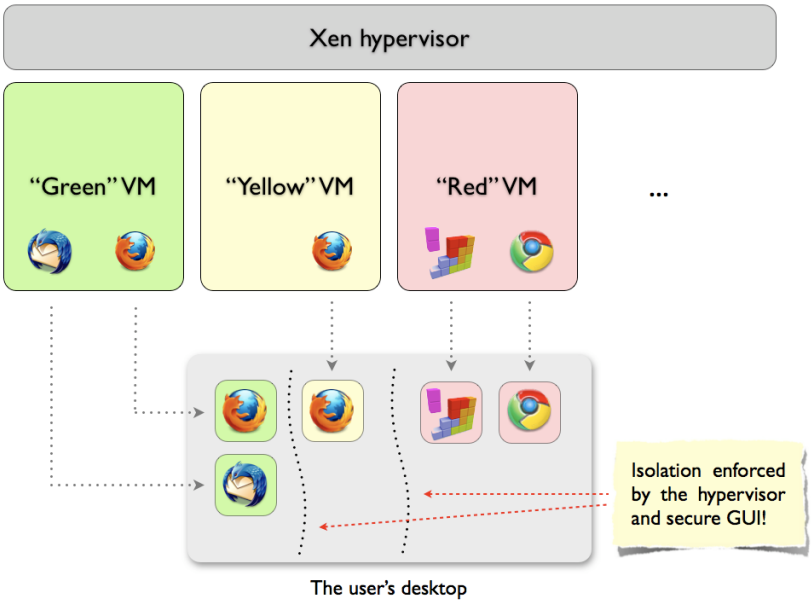


Figura 6.2: *Conceito de isolamento no Qubes-OS*

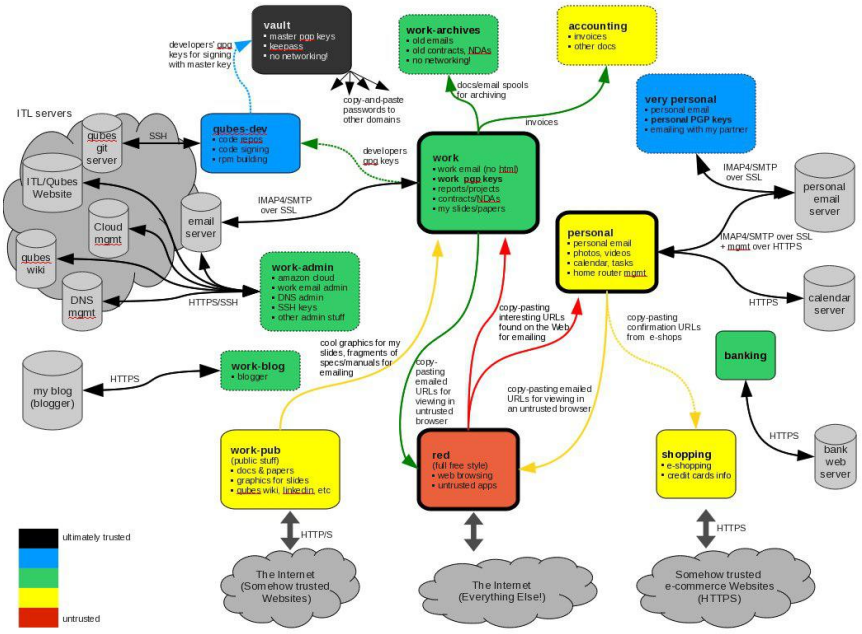


Figura 6.3: *Data flow dentro do Qubes-OS*

Capítulo 7

Conclusão

No presente trabalho, buscamos entender o estado atual da criptografia completa de disco (FDE), quais são os diferentes meios de consegui-la, estudar o nível atual da sua segurança e a possibilidade da sua utilização em SSDs.

Conseguimos identificar duas tecnologias principais de FDE: LUKS/*cryptsetup* para criptografia por *software* e SEDutil para criptografia de dispositivos que suportam a especificação OPAL 2.0.

Notamos uma razoável facilidade de utilização do LUKS em distribuições Linux comerciais, como *Ubuntu* e *Debian*, que já oferecem esse tipo de criptografia mesmo durante a própria instalação ou a partir do pacote *cryptsetup* disponível nos seus repositórios oficiais.

O SEDutil exige um maior conhecimento técnico do usuário como, por exemplo, a configuração da BIOS e familiaridade com a execução de comandos no terminal e é vulnerável a ataques de *HotPlug*.

Na análise de desempenho, constatou-se que nos dispositivos testados, houve uma redução entre 18,2% e 22,6% com suporte a AES e entre 17,28% e 23,73% sem suporte a AES na escrita no HDD e de 8% no SSD em relação à não utilização de criptografia. Considerou-se esses valores aceitáveis, dado que ambas as taxas de leitura se mantiveram iguais e os dados foram mantidos em sigilo.

A literatura nos mostra que tanto o *cryptsetup* quanto o OPAL são vulneráveis a algum tipo de ataque quando o dispositivo já foi desbloqueado na mesma sessão. Foram apresentadas diversas prevenções à esses ataques, incluindo a utilização correta do IOMMU que evita o ataque DMA e dá suporte à prevenção do ataque *Cold Boot* com o TreVisor e também o *boot* a partir de um *pendrive* para autenticar a máquina para o usuário, evitando o ataque *Evil Maid*. No caso do *Qubes-OS*, é utilizada a tecnologia TPM com o mesmo intuito.

Referências Bibliográficas

imi() The imitation game. https://www.imdb.com/title/tt2084970. Último acesso em 1/11/2016. Citado na pág. 3

wik() Registro do domínio wikileaks.org. https://www.godaddy.com/whois/domain.aspx?checkAvail=1&tmskey=&domain=wikileaks.org&prog_id=GoDaddy. Último acesso em 1/11/2016. Citado na pág. 3

AMD(2016) AMD. Amd i/o virtualization technology (iommu) specification. https://support.amd.com/TechDocs/48882_IOMMU.pdf, Dezembro 2016. Citado na pág. 39

Boileau(2006) Adam Boileau. Hit by a bus:physical access attacks with firewire. http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf, 2006. Citado na pág. 33

Boteanu e Fowler(2015) Daniel Boteanu e Kevvie Fowler. Bypassing self encrypting drives (sed) in enterprise environments. https://www.blackhat.com/docs/eu-15/materials/eu-15-Boteanu-Bypassing-Self-Encrypting-Drives-SED-In-Enterprise-Environments-wp.pdf, 2015. Citado na pág. 37

Böck(2009) Benjamin Böck. Firewire based physical security attacks on windows 7, efs and bitlocker. https://www.helpnetsecurity.com/dl/articles/windows7_firewire_physical_attacks.pdf, 2009. Citado na pág. 32

Carbone(2011) Carbone. An in-depth analysis of the cold boot attack. http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA545078, 2011. Citado na pág. 35

Choi *et al.***(2014)** Younsung Choi, Donghoon Lee, Woongryul Jeon e Dongho Won. Password-based single-file encryption and secure data deletion for solid-state drive. Em *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '14, páginas 5:1–5:7, New York, NY, USA. ACM. ISBN 978-1-4503-2644-5. doi: 10.1145/2557977.2558072. URL http://doi.acm.org/10.1145/2557977.2558072. Citado na pág. 13

cryptsetup() cryptsetup. cryptsetup - frequently asked questions. https://gitlab.com/cryptsetup/cryptsetup/wikis/FrequentlyAskedQuestions#5-security-aspects. Último acesso em 25/11/2017. Citado na pág. 13

Dornseif *et al.***(2004)** Maximillian Dornseif, Michael Becher e Christian N. Klein. 0wned by an ipod. https://cansecwest.com/core05/2005-firewire-cansecwest.pdf, 2004. Citado na pág. 32

Dornseif *et al.***(2005)** Maximillian Dornseif, Michael Becher e Christian N. Klein. Firewire. all your memory are belong to us. https://cansecwest.com/core05/2005-firewire-cansecwest.pdf, 2005. Citado na pág. 32

Gruhn e Müller(2013) Michael Gruhn e Tilo Müller. On the practicability of cold boot attacks. Em *Proceedings of the 2013 International Conference on Availability, Reliability and Security*, ARES '13, páginas 390–397, Washington, DC, USA. IEEE Computer Society. ISBN 978-0-7695-5008-4. doi: 10.1109/ARES.2013.52. URL https://www1.cs.fau.de/filepool/projects/coldboot/fares_coldboot.pdf. Citado na pág. 34

Gutmann(1996) Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. Em *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, SSYM'96, páginas 8–8, Berkeley, CA, USA. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1267569.1267577. Citado na pág. 34

Gutmann(2001) Peter Gutmann. Data remanence in semiconductor devices. Em *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, páginas 4–4, Berkeley, CA, USA. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1267612.1267616. Citado na pág. 34

Halderman *et al.*(2008) J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum e Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. Em *Proceedings of the 17th Conference on Security Symposium*, SS'08, páginas 45–60, Berkeley, CA, USA. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1496711.1496715. Citado na pág. 34

Herodotus(440 BC) Herodotus. The history of herodotus. http://classics.mit.edu/Herodotus/history.5.v.html, 440 BC. Último acesso em 1/11/2016. Citado na pág. 3

Intel(2016) Intel. Intel® virtualization technology for directed i/o, architecture specification. https://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf, Junho 2016. Citado na pág. 39

Intel(2016) Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C and 3D. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf, 2016. Citado na pág. 33, 39

Jaeger e Blalock(1990) Richard C. Jaeger e Travis N. Blalock. Quasi-static ram design for high performance operation at liquid nitrogen temperature. *Cryogenics*, 30(12):1030 – 1035. ISSN 0011-2275. doi: https://doi.org/10.1016/0011-2275(90)90203-O. URL http://www.sciencedirect.com/science/article/pii/001122759090203O. Citado na pág. 34

Karger e Herbert(1984) Paul A. Karger e Andrew J. Herbert. An augmented capability architecture to support lattice security and traceability of access. *1984 IEEE Symposium on Security and Privacy*, 00:2. ISSN 1540-7993. doi: doi.ieeecomputersociety.org/10.1109/SP.1984.10001. Citado na pág. 39

Kipp(2015) Scott Kipp. Will ssd replace hdd? http://www.ieee802.org/3/CU4HDDSG/public/sep15/Kipp_CU4HDDsg_01a_0915.pdf, 2015. Último acesso em 25/11/2017. Citado na pág. 5

Lindenlauf *et al.*(2015) Simon Lindenlauf, Hans Höfken e Marko Schuba. Cold boot attacks on ddr2 and ddr3 sdram. Em *Proceedings of the 2015 10th International Conference on*

Availability, Reliability and Security, ARES '15, páginas 287–292, Washington, DC, USA. IEEE Computer Society. ISBN 978-1-4673-6590-1. doi: 10.1109/ARES.2015.28. URL http://dx.doi.org/10.1109/ARES.2015.28. Citado na pág. 35

Liu *et al.*(2017) Chen Liu, Hoda A. Khouzani e Chengmo Yang. Erasucrypto a light-weight secure data deletion scheme for solid state drives. https://www.researchgate.net/publication/311883956_ErasuCrypto_A_Light-weight_Secure_Data_Deletion_Scheme_for_Solid_State_Drives, 2017. Citado na pág. 13

Markuze *et al.*(2016) Alex Markuze, Adam Morrison e Dan Tsafrir. True iommu protection from dma attacks: When copy is faster than zero copy. *SIGARCH Comput. Archit. News*, 44(2):249–262. ISSN 0163-5964. doi: 10.1145/2980024.2872379. URL http://doi.acm.org/10.1145/2980024.2872379. Citado na pág. 33, 42

Müller *et al.*(2011) Tilo Müller, Felix C. Freiling e Andreas Dewald. Tresor runs encryption securely outside ram. Em *USENIX Security Symposium*. URL https://www.usenix.org/legacy/event/sec11/tech/full_papers/Muller.pdf. Citado na pág. 40

Müller *et al.*(2012) Tilo Müller, Tobias Latzo e Felix Freiling. Self-Encrypting Disks pose Self-Decrypting Risks: How to break Hardware-based Full Disk Encryption. Relatório técnico. URL https://www1.informatik.uni-erlangen.de/filepool/projects/sed/seds-at-risks.pdf. Citado na pág. 36, 38

Müller *et al.*(2012) Tilo Müller, Benjamin Taubmann e Felix C. Freiling. Trevisor: Os-independent software-based full disk encryption secure against main memory attacks. Em *Proceedings of the 10th International Conference on Applied Cryptography and Network Security*, ACNS'12, páginas 66–83, Berlin, Heidelberg. Springer-Verlag. ISBN 978-3-642-31283-0. doi: 10.1007/978-3-642-31284-7_5. URL http://dx.doi.org/10.1007/978-3-642-31284-7_5. Citado na pág. 40, 41

Patterson(2011) Hennessy Patterson. Computer organization and design, fourth edition: The hardware/software interface, 2011. Citado na pág. 32

QubesOS(a) QubesOS. Qubes os: A reasonably secure operating system. https://www.qubes-os.org/, a. Último acesso em 27/11/2017. Citado na pág. 42

QubesOS(b) QubesOS. Anti evil maid (aem) | qubes os. https://www.qubes-os.org/doc/anti-evil-maid/, b. Último acesso em 27/11/2017. Citado na pág. 43

QubesOS(c) QubesOS. Wipe ram on shutdown. https://github.com/QubesOS/qubes-issues/issues/1562, c. Último acesso em 27/11/2017. Citado na pág. 43

Rutkowska(2009) Rutkowska. Evil maid goes after truecrypt! http://theinvisiblethings.blogspot.com.br/2009/10/evil-maid-goes-after-truecrypt.html, 2009. Citado na pág. 33

Rutkowska(2011) Joanna Rutkowska. Anti evil maid. https://blog.invisiblethings.org/2011/09/07/anti-evil-maid.html, 2011. Último acesso em 27/11/2017. Citado na pág. 43

Sacco e Ortega(2009) Anibal L Sacco e Alfredo A Ortega. Persistent bios infection: The early bird catches the worm. Em *Proceedings of the Annual CanSecWest Applied Security Conference, Vancouver, British Columbia, Canada. Core Security Technologies*. URL https://www.cansecwest.com/csw09/csw09-sacco-ortega.pdf. Citado na pág. 41

Shinagawa *et al.***(2009)** Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo e Kazuhiko Kato. Bitvisor: A thin hypervisor for enforcing i/o device security. Em *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, páginas 121–130, New York, NY, USA. ACM. ISBN 978-1-60558-375-4. doi: 10.1145/1508293.1508311. URL http://doi.acm.org/10.1145/1508293.1508311. Citado na pág. 40

Tails() Tails. Tails - protection against cold boot attacks. https://tails.boum.org/doc/advanced_topics/cold_boot_attacks/index.en.html. Último acesso em 27/11/2017. Citado na pág. 43

Trusted Computing Group, Incorporated(2012) Trusted Computing Group, Incorporated. TCG Storage Security Subsystem Class: Opal 2.00. http://www.trustedcomputinggroup.org/wp-content/uploads/TCG_Storage-Opal_SSC_v2.01_rev1.00.pdf, 2012. Citado na pág. 14

Yitbarek *et al.***(2017)** S. F. Yitbarek, M. T. Aga, R. Das e T. Austin. Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors. Em *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, páginas 313–324. doi: 10.1109/HPCA.2017.10. Citado na pág. 34