Aplicabilidade da Encriptação Especulativa em CPUs Multicore para Sistema de Arquivo no Espaço de Usuário

Vinícius Carlos Oliveira de Andrade
vcoandrade@inf.ufpr.br
Universidade Federal do Paraná
Curitiba, Brasil

Wagner Machado Nunan Zola
wagner@inf.ufpr.br
Universidade Federal do Paraná
Curitiba, Brasil

ABSTRACT

Due to the constantly growing need for security and efficiency in accessing stored data, there is also constant search for better storage media that bring this warranty. Previous research has led to the creation of libraries for parallel processing on GPU to perform speculative encryption of data. The efficiency of this method was demonstrated in its use in cryptographic file systems. However, the applicability and effectiveness of these methods using only CPUs remained an open problem. In order to test the feasibility of speculative cryptography in computing environments with CPUs only, this work presents a library version for speculative encryption with parallel processing on multicore processors and its adaptation to the EncFS++ file system in user space. The tests carried out show considerable increases in throughput in the most varied environments, both physical and virtual machines, using CPUs that may or may not support the AES-NI instruction set for encryption acceleration.

KEYWORDS

Criptografia Especulativa, Sistema de Arquivos Criptografados, WAESlib CPU, Modo CTR AES, Cifragem Especulativa em CPUs Multicore

1 INTRODUÇÃO

O enorme volume de dados gerados a cada ano, com estimativa de que cerca de 50 zettabytes tenham sido criados em 2020 [1], eleva a necessidade de segurança e eficiência no armazenamento dos mesmos. Uma forma de tentar sanar essa necessidade foi a criação de sistemas de arquivos criptografados dos mais variados tipos.

Muitos estudos buscam aumentar a eficiência dos algoritmos de criptografia ou dos sistemas de arquivo. Nesse contexto foi criada a WAESlib, que utiliza a tecnologia WAES para realizar encriptação especulativa de forma paralela em GPU. Com base na WAESlib foi implementado o sistema de arquivos EncFS++ que se utiliza de um esquema de janela deslizante para armazenar os contextos pré-processados utilizando a WAESlib. Apesar da existência de *delay* na transmissão de dados com o dispositivo e do perfil das requisições dificultar o trabalho em paralelo em GPU, devido a baixa carga de trabalho separados em eventos esparsos, foi demonstrada a viabilidade do sistema e que o esquema de janelas permite a obtenção de bons resultados.

Buscando mitigar alguns dos problemas encontrados com a versão em GPU, bem como gerar um sistema mais flexível foi desenvolvida a versão em CPU para a WAESlib e para o EncFS++, sendo o foco deste trabalho a demonstração da viabilidade do sistema para a realização de encriptação especulativa. Para isso foram feitos testes de desempenho sobre o sistema e verificado o comportamento e desempenho em diversos ambientes.

Na Seção 2 é encontrada a fundamentação teórica necessária para embasar os testes. A Seção 3 apresenta alguns trabalhos relacionados, mostrando estudos relativos a criptografia ou a sistemas de arquivos. Os testes e resultados são apresentados na Seção 4 e após isso uma conclusão é apresentada na Seção 5.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentadas algumas tecnologias necessárias para o bom desempenho do EncFS++ CPU. Iniciando com a tecnologia de encriptação utilizada pelo sistema, seguido de tecnologias de melhoria de processamento. Após, é mostrada a estrutura do sistema operacional Linux e do sistema EncFS, assim situando o ambiente de execução da WAESlib. Ao final da seção apresentamos as versões da WAESlib para CPU e o sistema de arquivos EncFS++.

2.1 AES e o modo CTR de encriptação

De modo a mitigar os problemas encontrados no DES e no 3DES o concurso do NIST exigia que os algoritmos concorrentes utilizassem blocos de 128 bit, em contraste com os 64 bits do DES, possuíssem chaves cujo tamanho fosse de pelo menos 128 bits e possuísse implementações eficientes tanto em software quanto em hardware. Seguindo as exigências do concurso o AES possui blocos de tamanho fixo de 128 bits e três tamanhos padrão para as chaves, 128, 192 e 256 bits.

O algoritmo utilizado pelo AES é dividido em diversas partes que são utilizadas de forma conjunta para se encontrar a cifra final. A primeira parte são as entradas, composta pela chave e pelo bloco que será encriptado. Internamente é encontrada outra parte do algoritmo composto pelas funções que são utilizadas para realizar o embaralhamento dos bits. E a terceira parte é o bloco de saída, composto por 128 bits, mesmo tamanho da entrada, no qual são armazenados os dados de saída embaralhados. De modo a garantir que os dados finais não possam ser utilizados para se chegar ao dado original ou à chave os processos internos são repetidos por um número de rodadas que varia de acordo com o tamanho da chave, sendo 10, 12 e 14 para chaves de tamanho 128, 192 e 256 respectivamente [2]. Sendo um algoritmo de cifragem simétrica o AES utiliza a mesma chave tanto para encriptar quanto para decriptar os dados [3], porém o processo de decriptação é um pouco diferente, de modo a ser o inverso do processo de encriptação.

Algoritmos de cifras de blocos são, de maneira geral, incapazes de diretamente realizar a cifragem de um número de bits maior que o tamanho do seu bloco. Sendo possuidor dessa limitação o AES não é capaz de cifrar mensagens maiores que 128 bits diretamente. Para que fosse possível a cifragem, de maneira segura, de dados maiores

foram criados os chamados modos de operação. Muito desses modos são mais antigos que o próprio AES e possuem características distintas.

Observando o conjunto de modos de operação existentes é possível identificar o modo CTR, que se destaca por ser muito mais novo e moderno quando comparado com o demais, uma vez que foi criado ignorando alguns dos fatores levados em consideração nos demais modos e que não faziam mais sentido. Quando comparado com os demais, sua definição é muito mais simples. O modo consiste na aplicação direta do AES em um conjunto de blocos de entrada chamados de contador [4]. Cada contador encriptado gera um bloco de saída o qual será utilizado em uma operação XOR com os dados em claro de modo a gerar a mensagem cifrada. Como a mensagem final é gerada a partir de uma operação XOR da mensagem em claro com um bloco cifrado, o processo de decriptação é idêntico ao processo de encriptação, apenas utilizando os blocos da mensagem cifrada no lugar dos blocos da mensagem em claro na entrada. Uma característica importante do modo CTR é que um contador e uma chave nunca devem ser utilizados juntos mais de uma vez, de modo a evitar riscos à segurança. [4]. A Figura 1 mostra o funcionamento do modo CTR.

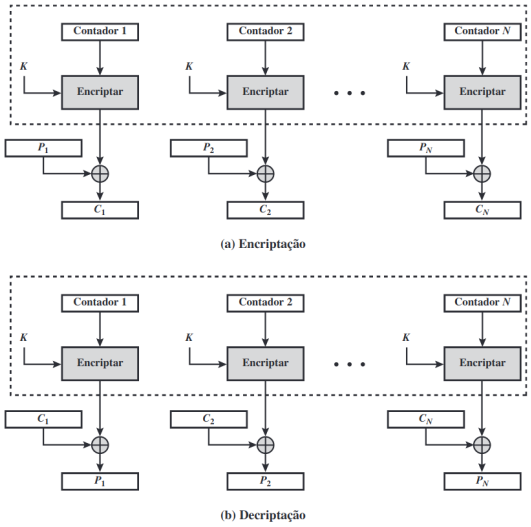

Figura 1: Funcionamento do modo CTR [5].

Para que a mensagem seja encriptada utilizando o modo CTR primeiramente são gerados um número de blocos cujo tamanho total seja o menor possível para acomodar a mensagem em claro por completo. Caso a mensagem não complete o último bloco, os bits sobressalentes devem ser descartados, não necessitando de *padding*.

Uma das principais características do modo CTR é o fato de os blocos cifrados serem totalmente independentes, não necessitando de informações dos blocos cifrados anteriores. Por esse motivo o modo CTR é completamente paralelizável, tanto na encriptação quando na decriptação uma vez que são processos idênticos.

A independência dos blocos cifrados, bem como o fato da mensagem final ser o resultado de um XOR de um desses blocos com um bloco da mensagem em claro permitem o surgimento de outra característica do modo, que é sua capacidade de pré-processamento dos blocos. O modo permite que os contadores sejam cifrados e armazenados até que seja necessário encriptar a mensagem final, quando os blocos armazenados são acessados e utilizados no XOR com a mensagem a ser encriptada. O pré-processamento, juntamente com a capacidade de ser processado em paralelo permite que o método CTR utilize de forma muito eficiente tanto o hardware quanto o software.

O modo CTR também possui a característica de possibilitar o acesso a posições aleatórias dos dados sem a necessidade de realizar a decriptação de todos os blocos anteriores [6]. Essa característica também é derivada da independência dos blocos no processo de encriptação e é bastante importante no contexto de armazenamento de dados utilizando este modo.

2.2 Extensões AVX e instruções AES-NI

De modo a tentar superar limitações físicas impostas ao hardware, projetistas se voltaram para a criação de CPUs capazes de operar em paralelo. Apesar de existirem muitas formas de implementação de paralelismo, uma forma bastante utilizada em CPUs é a chamada SIMD, sigla para *single instruction multiple data*. O modelo de paralelismo SIMD possui como característica a utilização de uma mesma instrução em conjuntos distintos de dados de maneira simultânea.

Um exemplo importante de paralelismo SIMD são as extensões AVX que foram introduzidas pela Intel em seus processadores em 2008. A extensão é composta por um conjunto de instruções que realizam processamento paralelo de categoria SIMD [7] e foi projetada para substituir as instruções SIMD existentes anteriormente. Por ser apresentada como um substituto das tecnologias que já eram usadas, o AVX faz uso de registradores maiores do que era possível anteriormente, utilizando 256 bits, com suporte para ampliação no futuro [7], em suas versões AVX1 e AVX2, e AVX-512, de 512 bits, lançada em 2017 [8].

Para que o AVX possa ser utilizado, são implementados em hardware 16 registradores de 256 bits, ou 512 no caso do AVX-512. Além disso, também existe um registrador de controle de 32 bits [7]. Diferente das tecnologias anteriores, o AVX não necessita que a memória esteja alinhada para que seja carregada nos registradores, porém o não alinhamento pode acarretar em perda de desempenho. Desde sua primeira apresentação, as instruções AVX já mostram um ganho de desempenho duas vezes maior quando comparado com a tecnologia SSE anterior [7].

Outra tecnologia importante relacionada ao ganho de desempenho no processamento de encriptação utilizando o algoritmo AES são as chamadas instruções AES-NI, sigla do inglês que significa *AES new instructions*. A necessidade de se realizar tarefas com encriptações e decriptações levou a Intel, em 2010, a criar as instruções que permitem o processamento do AES de maneira muito rápida e segura [9].

O AES-NI possui seis instruções que são divididas em dois conjuntos, sendo 4 responsáveis pela encriptação e decriptação e duas responsáveis pela expansão da chave, realizada internamente pelo algoritmo AES. As instruções foram projetadas para funcionarem em diversos modos do algoritmo AES, incluindo todos os tamanhos de chave e algumas versões não padrão [9]. Por utilizar um número reduzido de instruções para representar o algoritmo inteiro, estas

instruções permitem uma implementação simples da cifragem com AES, evitando erros no código.

Quando implementado em software, as versões mais eficientes do AES são aquelas que se utilizam de buscas em tabelas pré-processadas. Quando comparada com as versões em software mais rápidas, que utilizam tabelas, a versão em hardware do AES-NI apresenta ganhos entre três e quatro vezes, para modos não paralelizáveis como o CBC, podendo chegar em até 10 vezes, para modos paralelizáveis como o modo CTR [9].

2.3 Estrutura do Linux

O Linux é um sistema operacional e, como tal, necessita realizar duas funções que são obrigatórias em qualquer sistema operacional: gerenciar o hardware e oferecer um conjunto abstrato de interfaces simples aos programadores [10]. Para que seja capaz de realizar tais funções o Linux é dividido em diversas camadas de abstração, desde o usuário até o hardware.

Este modelo de operação, juntamente com as interfaces entre as camadas, permite que desenvolvedores e usuários vejam as camadas mais complexas e heterogêneas, mais próximas do hardware, como chamadas padronizadas mais simples. Essa visão é chamada de sistema operacional como máquina estendida [10].

Outro detalhe importante é a existência dos modos de operação executando instruções do núcleo, também chamado de modo *kernel*, ou executando instruções do usuário. Enquanto o modo usuário é limitado, o modo núcleo é privilegiado e possui total acesso e controle sobre o hardware. O modo usuário é responsável por realizar a execução de aplicações do usuário, bem como controlar os recursos encontrados no espaço de usuário. O espaço de usuário é um subconjunto dos recursos disponíveis na máquina [11]. O modo usuário pode realizar algumas chamadas de sistema, no entanto, além de não possuir acesso direto ao hardware também não possui acesso aos recursos fora do espaço de usuário. Por outro lado o modo núcleo, por ser privilegiado e possuir todo acesso ao hardware é o responsável por realizar o acesso ao hardware quando requisitado pelos usuários e disponibiliza os recursos para os mesmos [10][12]. A utilização dos dois modos é necessária para evitar problemas de sincronização, dois ou mais usuários requisitando o mesmo recursos ao mesmo tempo, ou problemas de acesso indevido, uma aplicação de usuário alterando algum local de memória que não deveria sofrer modificações.

As operações em modo núcleo são realizadas pelo núcleo do sistema operacional, que é a parte mais interna e mais próxima do hardware. Com o objetivo de se tornar mais versátil, o núcleo do Linux, que é um núcleo monolítico, utiliza o conceito de módulos. Módulos são blocos de código que podem ser carregados enquanto o sistema está em execução [10] e permite ao Linux se adaptar fácil e rapidamente para ser utilizado com diferentes hardwares, permitindo inclusive a troca de alguns componentes de hardware sem a necessidade de realizar paradas nos sistema.

Além dos módulos existem também os chamados subsistemas, que são responsáveis por implementar uma função do sistema operacional e são compostos de um ou mais módulos. A utilização de módulos e subsistemas permite o funcionamento transparente das mais diversas aplicações de usuário, devido a padronização das chamadas independente do hardware utilizado abaixo. O núcleo do sistema pode ser entendido como um conjunto composto de uma parte monolítica, módulos e subsistemas.

Um subsistema indispensável para um bom funcionamento de um sistema operacional Linux é o subsistema chamado VFS, sigla do inglês para sistema de arquivos virtual. Este subsistema é responsável por gerenciar os sistemas de arquivos e sua comunicação com o hardware. O VFS apresenta interfaces padronizadas para que os programas executados em modo usuário possam realizar chamadas [11]. As abstrações realizadas pelo VFS permitem que vários tipos diferentes de sistemas de arquivos possam coexistir, sendo armazenados nos mais variados tipos de mídia [10], por esse motivo o VFS é o ponto central para a criação de sistemas de arquivos criptografados em espaço de usuário. A Figura 2 mostra de maneira abstrata o conteúdo do VFS.

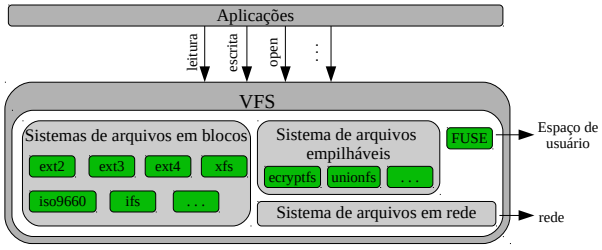

Figura 2: Componentes de um VFS, baseado em [13]

Na figura é possível ver diversos módulos responsáveis pela comunicação com vários outros sistemas, seja um sistema de arquivo ou um tipo de hardware ou algum outro sistema distinto. Os sistemas de arquivos em blocos formam um conjunto de módulos importantes, que é como são conhecidos os sistemas de arquivos mais comumente utilizados pelos usuários e que funcionam sobre o hardware. Outro conjunto de módulos importantes são os sistemas de arquivos empilháveis, que não funcionam diretamente sobre o hardware mas sim sobre outro sistema de arquivos.

Outro módulo importante que se observa na Figura 2 é o chamado FUSE, sigla do inglês para *filesystem in userspace* se traduz para sistemas de arquivo no espaço do usuário. Este módulo permite que sejam criados sistemas de arquivos em espaço de usuário.

De maneira simplificada o módulo FUSE pode ser visto como um protocolo cliente servidor, onde o núcleo, utilizando o módulo FUSE, é o cliente e a aplicação criada pelo desenvolvedor do sistema de arquivos é o servidor, sendo essa aplicação executada em espaço de usuário. A Figura 3 mostra o funcionamento de um sistema utilizando FUSE.

2.4 EncFS

Criado em 2003 o EncFS é um sistema de arquivos criptografado, cuja execução é realizada em espaço de usuário através do módulo FUSE. Foi criado com o objetivo de suprir uma demanda criada pela ausência de sistemas de arquivos criptografados que não fossem baseados em NFS [15] ou em espaço de núcleo [16].

O EncFS utiliza dois modos de encriptação diferentes para criptografar partes diferentes do arquivo. Para a encriptação dos nomes dos arquivos e de blocos incompletos, blocos finais cujo tamanho

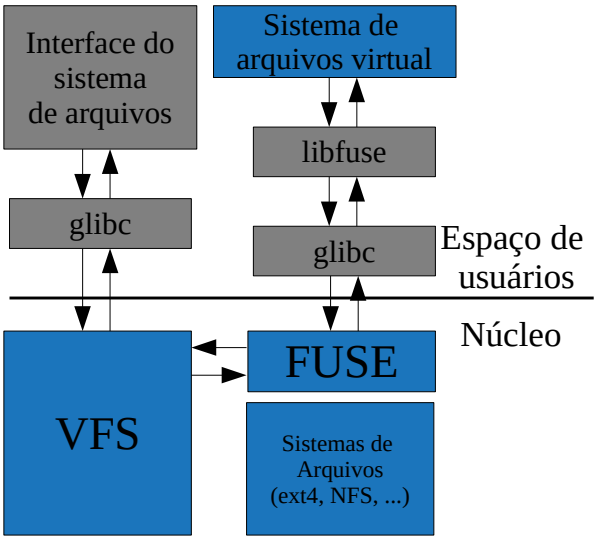

Figura 3: Exemplo de funcionamento de um sistema de arquivos utilizando FUSE, baseado em [14].

é menor que o tamanho máximo de um bloco, é utilizado o modo CFB. Para a encriptação dos dados dos arquivos é utilizado o modo de operação CBC, que é utilizado para encriptar blocos completos de tamanho 4KiB. De modo a facilitar o acesso aleatório aos dados, cada bloco possui um IV determinístico. Quando o sistema de arquivos é criado ele solicita uma senha que é utilizada na encriptação e decriptação da senha de volume, que é utilizada nos processos envolvendo os arquivos, permitindo que os usuários alterem a senha sem a necessidade de encriptar novamente todo o sistema de arquivos. Para realizar a montagem de um sistema de arquivos já existente o usuário precisa informar a senha correta.

2.5 WAESlib

A biblioteca WAESlib foi o resultado da implementação do WAES para o funcionamento em GPUs utilizando a plataforma CUDA. A biblioteca buscava apresentar uma interface que permitisse a utilização de encriptação especulativa combinado com o processamento paralelo em GPU. Embora projetada para funcionar em GPU, as técnicas utilizadas pela biblioteca não são dependentes desta tecnologia, permitindo que a biblioteca seja alterada para funcionar em CPU. A biblioteca apresenta como interface um conjunto de seis funções, WAES_init, WAES_setkey, WAES_ctx, WAES_encrypt, WAES_decrypt e WAES_finish.

As funções WAES_init e WAES_finish servem para inicializar e finalizar todos os processos e threads utilizados na execução, incluindo também alocação e liberação de memória. Por sua vez a função WAES_setkey prepara e armazena a chave que será utilizada nas encriptações de modo a minimizar o overhead nos processos futuros de encriptação. As funções WAES_encrypt e WAES_decrypt são idênticas e realizam a operação XOR entre o buffer e o contexto informado. Por último a função WAES_ctx é a função responsável por agendar e iniciar o processo de preparação dos contextos que serão utilizados futuramente nos processos de encriptação e decriptação.

2.6 EncFS++

O EncFS++ é o resultado da busca por um sistema de arquivos criptografado que utilizasse as técnicas de processamento paralelo e encriptação especulativa, utilizando a WAESlib. O EncFS++ foi criado com base no EncFS e utiliza a WAESlib para o gerenciamento dos contextos de encriptação. Sua implementação realiza diversos ajustes necessários para o funcionamento do sistema de arquivos com o modo CTR, de maneira a suprir as exigências necessárias para o correto e seguro funcionamento do modo de operação, e para a utilização de encriptação especulativa.

A principal exigência para o funcionamento do modo CTR é a de que um par (contador, chave) nunca seja utilizado mais de uma vez. A solução escolhida foi a geração determinística dos contadores onde eles são divididos em duas partes, sendo uma global e variando entre arquivos e outra local do CTR e variando para cada bloco encriptado. O tamanho máximo para um arquivo varia com a quantidade de bits reservados para o contador.

Para o correto e eficiente funcionamento da encriptação especulativa, o EncFS++ faz uso de um conjunto de contextos organizados em filas circulares. Quando se deseja encriptar ou decriptar um bloco de um arquivo, são gerados contextos que são armazenados em uma fila. Quando o contexto é utilizado sua posição na fila é liberada e utilizada para armazenar o próximo contexto. Quando o próximo bloco do arquivos for acessado seu contexto já foi pré-processado e está na fila, pronto para ser utilizado. Este conceito permite que o contexto sempre seja processado antes que seja necessário seu uso, fazendo pleno uso da encriptação especulativa. Para acessos a um bloco aleatório de um arquivo é possível calcular qual posição da fila armazenará o contexto necessário e seguir o mesmo processo descrito anteriormente. Esta implementação também permite alteração na posição da fila, incluindo os casos em que os contextos podem ou não ser reaproveitados.

2.7 WAESlib CPU e EncFS++ CPU

Com base nas versões já existentes da WAESlib GPU e do EncFS++, que faz uso dela, foi criada a biblioteca WAESlib versão CPU e acoplado ao sistema de arquivos EncFS++ para que o mesmo funcione apenas utilizando encriptação em CPU.

A versão em CPU possui a vantagem de ser mais flexível quando comparada com a versão em GPU. Alguns exemplos são ambientes virtualizados, ou que não possuem hardware específico para o processamento da encriptação. Enquanto a versão em GPU necessita de hardware específico para funcionar e de configurações complexas para que possa ser utilizado em ambientes virtualizados, a versão em CPU funciona da mesma forma em todos os ambientes.

3 TRABALHOS RELACIONADOS

O aumento na eficiência tanto dos algoritmos de criptografia quanto dos sistemas de arquivos é o objetivo de muitas buscas e são apresentados em vários trabalhos. Alguns trabalhos [17] apresentam variáveis que afetam a implementação paralela do algoritmo AES em GPU e que, até certo nível, podem ser ampliados para implementações paralelas em CPU.

Outros estudos importantes [18] verificaram a implementação do AES em CPUs de baixa capacidade, e em em sistemas [19] intermediários e GPUs. Outros estudos [20] comparam as implementações em software e implementaçoes em GPU com implementações aceleradas com AES-NI.

A procura por melhor desempenho levou ao surgimento da WAES [21], que consiste em uma implementação do AES em GPU que faz uso do pré-processamento do modo CTR para a realização de encriptação especulativa. Esta técnica foi posteriormente utilizada para a criação de um sistema de arquivos.

O número de estudos relativos a sistemas de arquivos também é bastante grande. Alguns estudos [22] comparam sistemas de arquivos em espaço de usuário implementados em linguagens diferentes. Outros trabalhos [23] [24] analisaram o comportamento dos sistemas de arquivos sobre diferentes cargas.

Também foram realizados estudos buscando criar sistemas de arquivos acelerados utilizando GPU [25]. Posteriormente esses sistemas acelerados puderam ser expandidos para que também trouxessem segurança para os arquivos armazenados [26].

4 EXPERIMENTOS E RESULTADOS

Nos experimentos foi utilizada versão CPU do sistema de arquivos EncFS++. Seguindo a linha inicial de testes realizados para o sistema de arquivos versão em GPU foram realizados testes com a ferramenta *Filebench* com as cargas *fileserver.f* e *webserver.f*. As cargas apresentam comportamentos bem distintos, o que permite uma melhor análise sobre o comportamento do sistema de arquivos em um ambiente real. Os testes simulam o acesso aos arquivos em um período de um minuto. Como o critério de parada do teste é o tempo de execução o número de repetições varia conforme o desempenho de cada ambiente de teste, porém mesmo nos ambientes com menor desempenho são realizadas dezenas de milhares de repetições.

Foram realizados testes em ambientes físicos e virtuais para que fosse possível comparar o comportamento do sistema em cada um deles. Como mudanças de contexto e acesso aos dispositivos de armazenamento em ambientes virtuais podem gerar chamadas adicionais no hipervisor não é claro se o comportamento será o mesmo em ambos os ambientes, sendo necessário o teste para verificar a utilização do sistema de arquivos em ambientes virtuais.

Para os testes em ambiente físico foi utilizado um computador um Intel i7-11700F, com 2.50GHz, 8 Cores e 16 *threads*, 16GB de memória e dois dispositivos de armazenamento, sendo um NVMe de alta vazão ADATA SX6000LNP e um SSD Kingston SA400S37.

Nos testes virtuais foi utilizada uma máquina virtual sobre um virtualizador VMware ESXi 6.7. A máquina virtual possui 8 CPUs virtuais, cada uma com uma *thread* e ligada a um *core* do processador Xeon Gold 5218 de 2,3GHz, 16GB de memória RAM e dois discos virtuais criados sobre HDs e SSDs físicos.

Para todos os testes, tanto em ambiente físico quanto em virtual, foi utilizado o sistema operacional Ubuntu 18.04, com a biblioteca libfuse versão 2.9.7 e openssl versão 1.1.1. Como o EncFS++ é um sistema de arquivos empilhado foi utilizado o sistema de arquivos ext4 como base em todos os testes.

Uma vez que a versão CPU do sistema de arquivos pode ser utilizada em ambientes com maiores restrições de hardware também se fez necessário a realização de testes que simulassem ambientes com hardware mais simples, ou que não possuíssem hardware específico para o processamento de criptografia. Dessa maneira foram realizados testes com o AES-NI habilitado e desabilitado.

Para cada versão dos testes também foi realizado um teste utilizando o sistema de arquivos EncFS, base de criação do EncFS++, para que seja realizada a comparação. Como todo o processamento, tanto dos sistemas de arquivos quanto dos testes, é realizado em CPU foram feitos testes com o *Filebench* utilizando entre 1 e 10 *threads*.

Não foram feitas comparações entre os testes das versões em CPU e em GPU uma vez que os desafios abordados foram diferentes para cada um dos sistemas. A comparação entre os sistemas também não seria justa uma vez que os ambientes de teste são específicos e bastante diferentes em cada um dos trabalhos.

A Figura 4 apresenta o resultado do primeiro teste. O teste foi realizado utilizando a carga fileserver.f em um ambiente com físico com AES-NI habilitado e em um dispositivo de armazenamento NVMe de alta vazão.

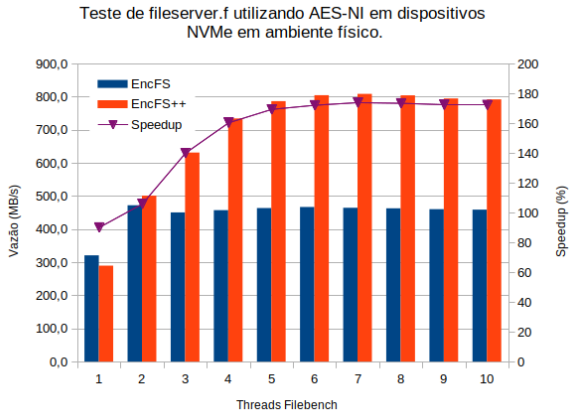

Figura 4: Teste do filebench utilizando carga fileserver.f, com AES-NI habilitado em ambiente físico sobre um dispositivo NVMe de alta vazão.

No eixo x é apresentado o número de threads utilizadas pelo *Filebench*, também chamadas de *threads* clientes. É possível ver que enquanto o EncFS logo estabiliza em um valor, quando possui apenas dois clientes, o EncFS++ demora mais a estabilizar, com 7 clientes, com uma vazão muito mais alta. Também é possível notar que a vazão cai para valores maiores que 8. Isto se dá devido a sobrecarga do processador que possui 8 núcleos. Embora o pico tenha ocorrido para 7 clientes o valor da vazão para 6, 7 e 8 clientes difere em menos de 0,5%.

Para a comparação direta entre ambos os sistemas é possível ver um *speedup* de 174% entre eles. Também é possível calcular o *speedup* entre as vazões máximas de cada um deles. O cálculo para os máximos mostra 171% de aumento, com picos de 808,4MB/s e 472,2MB/s. Para os testes utilizando a carga webserver.f o *speedup* entre os máximos é um pouco menor, 126% para vazões de 592,3MB/s e 470,1MB/s. Enquanto o valor para o EncFS fica praticamente igual, o mesmo não acontece com o EncFS++. Isso se

deve ao teste com webserver.f não permitir o uso de encriptação especulativa de forma tão eficiente quanto o teste com fileserver.f.

A Figura 5 apresenta um teste semelhante ao anterior, porém com o AES-NI desativado, simulando um ambiente com menor capacidade de hardware.

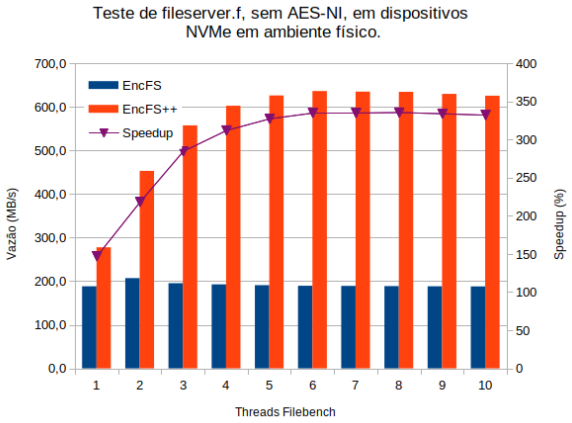

Figura 5: Teste do filebench utilizando carga fileserver.f, sem AES-NI habilitado em ambiente físico sobre um dispositivo NVMe de alta vazão.

Como é possível ver, o comportamento do sistema de arquivos EncFS se mantém o mesmo com ou sem AES-NI, possuindo um máximo para 2 clientes e estabilizando posteriormente, porém sua vazão é menos da metade do teste original, com uma queda de 56,13%. De forma similar o comportamento do EncFS++ se manteve igual em ambos os testes, com pico em 6 clientes e vazão para 6, 7 e 8 clientes com menos de 0,5%, porém sua perda de desempenho foi muito menos acentuada, com queda de 15,52%. Com o cálculo dos *speedups* são encontrados valores de até 336%. Quando são comparados os valores máximos o número cai para 329%, com vazões 682,9MB/s e 207,1MB/s. Para o webserver.f um cenário similar ao teste anterior ocorre, onde o EncFS mantém uma vazão próxima, 208,3MB/s, enquanto o EncFS++ apresenta uma queda proporcional maior, com 524,4MB/s, resultando em um *speedup* de 251%.

As Figuras 6 e 7 mostram os resultados dos mesmos testes sobre um dispositivos SSD.

Como é possível ver através dos gráficos não existem mudanças significativas no comportamento as aplicações mesmo com a diferença na velocidade de acesso dos dispositivos.

A falta de alteração não é apenas no comportamento mas também na vazão. Para todos os testes realizados a diferença entre a vazão máxima para os testes sobre NVMe e sobre SSD é menor que 5%, sendo a maior parte menor que 1%.

As Figuras 8 e 9 apresentam os gráficos relativos aos testes em ambientes virtualizados com os discos virtuais criados sobre SSDs.

Os valores máximos da vazão de ambos os sistemas caem para aproximadamente a metade dos valores encontrados no ambiente físico. Essa queda se dá pelo aumento do *overhead* que as instruções virtuais sofrem, tanto na parte de processamento quando na de acesso aos dados em disco.

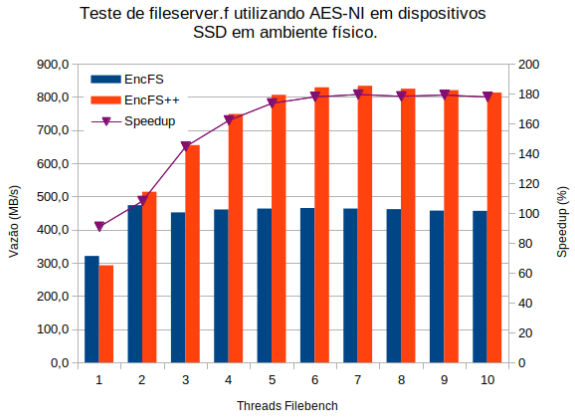

Figura 6: Teste do filebench utilizando carga fileserver.f, com AES-NI habilitado em ambiente físico sobre um dispositivo SSD.

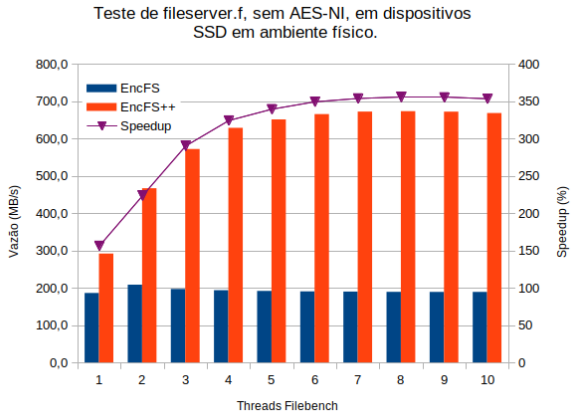

Figura 7: Teste do filebench utilizando carga fileserver.f, sem AES-NI habilitado em ambiente físico sobre um dispositivo SSD.

Para o EncFS++, embora exista uma variação percentual maior nos valores mais altos para a vazão o comportamento ainda está de acordo com o encontrado no ambiente físico, com máximo em 6 clientes e valores próximos para 6, 7 e 8.

Para o EncFS o comportamento é um pouco diferente, não sendo mais o máximo com dois clientes, mas sim com 7. Com instruções mais demoradas o sistema ganha mais desempenho quando atende mais cliente, otimizando o tempo de espera das requisições anteriores.

Pelo gráfico é possível observar *speedups* de até 188%. Quando são comparadas as vazões máximas dos dois sistemas é verificado um *speedup* de 183%, para vazões de 395,1 MB/s e 215,1MB/s. Assim como no ambiente físico o ganho é menor nos testes com webserver.f, sendo 270,3MB/s e 218MB/s resultando em um *speedup* de 124%.

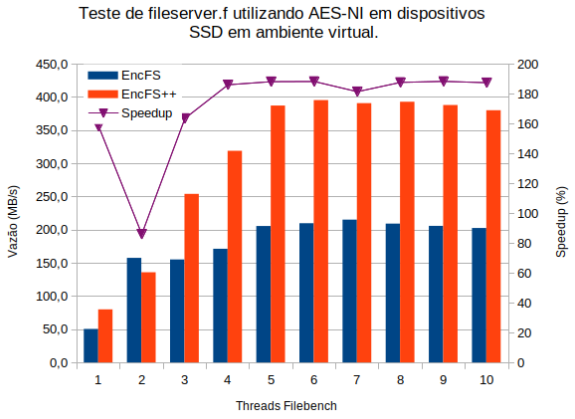

Figura 8: Teste do filebench utilizando carga fileserver.f, com AES-NI habilitado em ambiente virtual sobre um dispositivo SSD.

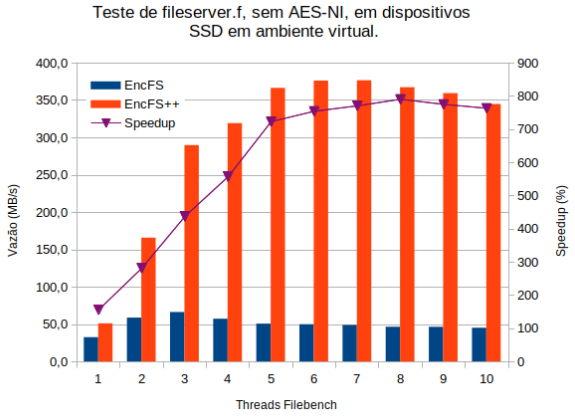

Figura 9: Teste do filebench utilizando carga fileserver.f, sem AES-NI habilitado em ambiente virtual sobre um dispositivo SSD.

Quando o AES-NI é desabilitado no ambiente virtual são encontradas reações opostas dos sistemas EncFS e EncFS++ quando comparados com os testes físicos. Enquanto em todos os testes a vazão diminui quando o AES-NI é desabilitado o EncFS++ possui a menor queda dentre todos os testes em todos os ambientes, 4,73% entre as vazões máximas das versões com e sem AES-NI. Em contrapartida a mesma mudança quando aplicada ao EncFS resulta na maior perda de desempenho de todos os testes, com 69,72%. Desta maneira são encontrados os maiores *speedups* entre as duas versões nos ambientes virtualizados e sem AES-NI, totalizando 569% quando comparadas as vazões máximas de 376,4MB/s e 66,1MB/s. Para o webserver.f temos as vazões de 252,1MB/s e 55,4MB/s, resultando em um *speedup* de 455%.

Embora o comportamento relativo ao número de clientes tenda a se manter o mesmo a variação percentual dos valores para o EncFS se torna bastante perceptível devido aos baixos valores absolutos.

Nos gráficos das Figuras 10 e 11 são mostrados os resultados dos testes da carga fileserver.f em um disco virtual criado sobre HDDs.

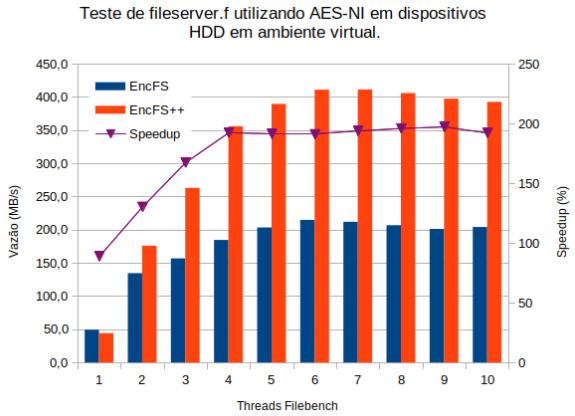

Figura 10: Teste do filebench utilizando carga fileserver.f, com AES-NI habilitado em ambiente virtual sobre um dispositivo HDD.

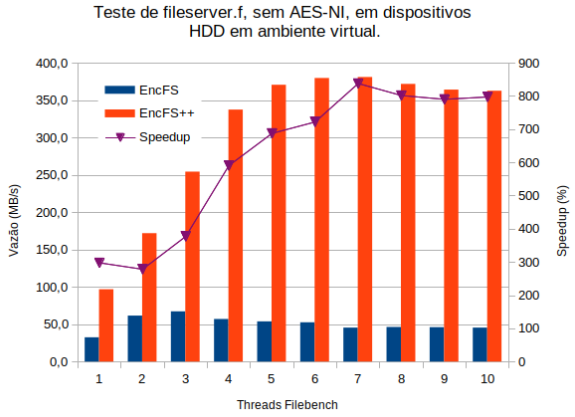

Figura 11: Teste do filebench utilizando carga fileserver.f, sem AES-NI habilitado em ambiente virtual sobre um dispositivo HDD.

Assim como no ambiente físico a mudança de dispositivo de armazenamento não afeta o comportamento do sistema, mesmo com velocidades de acesso distintas os valores das vazões diferem em menos de 5%. É importante verificar que no teste sem AES-NI apesar da variação absoluta ser pequena a variação percentual é grande, devido aos baixos valores, o que leva a variações muito grandes no *speedup*, resultando em valores diretos muito maiores do que os encontrados quando calculado comparados os valores máximos.

Tabela 1: Consolidação das vazões em MByte/s para ambiente físico utilizando AES-NI

	NVMe de alta vazão		SSD	
	fileserver.f	webserver.f	fileserver.f	webserver.f
EncFS++	808,4	592,3	833,7	592
EncFS	472,1	470,1	474,1	466,2
Speedup	171,23%	125,99%	175,85%	126,98%

Tabela 2: Consolidação das vazões em MByte/s para ambiente virtual utilizando AES-NI

	HDD		SSD	
	fileserver.f	webserver.f	fileserver.f	webserver.f
EncFS++	411,2	278,7	395,1	270,3
EncFS	214,7	213,6	215,1	218
Speedup	191,52%	130,48%	183,68%	123,99%

Tabela 3: Consolidação das vazões em MByte/s para ambiente físico sem utilizar AES-NI

	NVMe de alta vazão		SSD	
	fileserver.f	webserver.f	fileserver.f	webserver.f
EncFS++	682,9	524,4	672,7	522
EncFS	207,1	208,3	208,3	208,3
Speedup	329,74%	251,75%	322,95%	250,60%

Tabela 4: Consolidação das vazões em MByte/s para ambiente virtual sem utilizar AES-NI

	HDD		SSD	
	fileserver.f	webserver.f	fileserver.f	webserver.f
EncFS++	380,9	262,3	376,4	252,1
EncFS	67,2	77,8	66,1	55,4
Speedup	566,82%	337,15%	569,44%	455,05%

Na versão com AES-NI ativo são encontradas vazões de 411,2MB/s e 214,7MB/s, *speedup* de 191%, no para o fileserver.f e 278,7MB/s e 213,6MB/s, *speedup* de 130%, para o webserver.f. Já com o AES-NI desativado temos 380,9MB/s e 67,2MB/s, *speedup* de 566%, para o fileserver.f e 262,3MB/s e 77,8MB/s, *speedup* de 337%, para o webserver.f.

As tabelas a seguir apresentam os valores consolidados para as vazões, em MB/s, e dos *speedups*, para os ambientes de teste apresentados.

5 CONCLUSÃO

Os experimentos realizados mostram a viabilidade de se utilizar sistemas de arquivos encriptados baseados em encriptação especulativa e paralelismo em CPU. Não só o sistema se apresenta de forma versátil como também confiável, mostrando o comportamento esperado nos mais diversos ambientes.

Além de sua versatilidade e confiabilidade, podendo ser utilizado nos mais diversos ambientes, a versão em CPU também apresenta grande aumento no desempenho quando comparado com sistemas similares que não utilizam a encriptação especulativa, com *speedups* entre 250% e 569% em diversos ambientes, incluindo ambientes virtualizados ou sem hardware específico para processamento de criptografia.

REFERÊNCIAS

[1] John Rydning David Reinsel, John Gantz. The digitization of the world - from edge to core. Technical Report US44413318, Seagate, 2018.
[2] Specification for the advanced encryption standard (AES). NIST - Federal Information Processin Standards Publication 197, 2001.
[3] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.
[4] Recommendation for block cipher modes of operation. NIST - Federal Information Processing Standards Publication SP 800-38A, 2001.
[5] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson, 2017.
[6] Phillip Rogaway. Evaluation of some blockciphermodes of operation. Technical report, Dept. of Computer Science - University of California, Davis, California, USA, 2011.
[7] Chris Lomont. Introduction to intel advanced vector extensions. June 2011.
[8] Intel advanced vector extensions 512 (intel AVX-512). Intel Corporation, 2020.
[9] Shay Gueron. Intel advanced encryption standard (AES) new instructions set. Technical Report 323641-001, Intel, 2010.
[10] Andrew Tanenbaum and Herbert Bos. *Modern Operating Systems*. Pearson, 2015.
[11] Robert Love. *Linux Kernel Development*. Developer's Library, 2010.
[12] Daniel P. Bovet and Marco Cesati. *Understanding the Linux Kernel*. O'Reilly, 2005.
[13] Werner Fischer. Linux storage stack diagram. https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram, 2018.
[14] Mohammad Banikazemi, David Daly, and Bulent Abali. Sysman: A virtual file system for managing clusters. In *Proceedings of the 22nd Large Installation System Administration Conference (LISA '08)*, San Diego, CA, USA, November 2008.
[15] Internet Engineering Task Force (IETF). Network file system (NFS) version 4 protocol. https://tools.ietf.org/html/rfc7530, 2015.
[16] Valient Gough. EncFS: An encrypted filesystem for FUSE. https://github.com/vgough/encfs, 2020.
[17] Q. Li, C. Zhong, K. Zhao, X. Mei, and X. Chu. Implementation and analysis of aes encryption on GPU. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 843–848, 2012. doi: 10.1109/HPCC.2012.119.
[18] Dag Arne Osvik, Joppe W. Bos, Deian Stefan, and David Canright. Fast software AES encryption. FSE'10, page 75–93, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3642138578.
[19] Naoki Nishikawa, Keisuke Iwai, and Takakazu Kurokawa. High-performance symmetric block ciphers on multicore CPU and GPUs. *International Journal of Networking and Computing*, 2(2):251–268, 2012. doi: 10.15803/ijnc.2.2_251.
[20] Guang liang Guo, Quan Qian, and Rui Zhang. Different implementations of AES cryptographic algorithm. *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 1848–1853, 2015.
[21] Wagner M. Nunan Zola and L. C. E. de Bona. Parallel speculative encryption of multiple AES contexts on GPUs. *2012 Innovative Parallel Computing (InPar)*, pages 1–9, 2012.
[22] Aditya Rajgarhia and Ashish Gehani. Performance and extension of user space file systems. pages 206–213, 01 2010. doi: 10.1145/1774088.1774130.
[23] Vasily Tarasov, Abhishek Gupta, Kumar Sourav, Sagar Trehan, and Erez Zadok. Terra incognita: On the practicality of user-space file systems. In *7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15)*, Santa Clara, CA, July 2015. USENIX Association.
[24] Bharath Kumar Reddy Vangoor, Prafful Agarwal, Manu Mathew, Arun Ramachandran, Swaminathan Sivaraman, Vasily Tarasov, and Erez Zadok. Performance and resource utilization of fuse user-space file systems. *ACM Trans. Storage*, 15(2), May 2019. ISSN 1553-3077. doi: 10.1145/3310148. URL https://doi.org/10.1145/3310148.
[25] Chien-Kai Tseng, Shang-Chieh Lin, and Y. Hsu. A file system using GPU-accelerated file-wise reliability scheme. In *Proceedings of the International Conference on Parallel and Distributed Processing Tech- niques and Applications (PDPTA 12)*, page 32–38, Las Vegas, 2012. CSREA Press.
[26] Shang-Chieh Lin, Yu-Cheng Liao, and Yarsun Hsu. A reliable and secure GPU-assisted file system. *Algorithms and Architectures for Parallel Processing, Lecture Notes in Computer Science*, 8630:71–84, 08 2014. doi: 10.1007/978-3-319-11197-1-6.