Utilização de Criptografia em Modo CTR Aplicada ao Armazenamento de Arquivos em Nuvem

Vandeir Eduardo[2], Luis Carlos E. de Bona[1], Wagner M. Nunan Zola[1]

[1] Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

[2]Departamento de Sistemas e Computação – Universidade Regional de Blumenau (FURB)
Rua Antônio da Veiga, 140 – 89.030-903 – Blumenau – SC – Brasil

vanedu@furb.br, bona@inf.ufpr.br, wagner@inf.ufpr.br

Abstract. *This work describes an innovative approach to using CTR mode of operation applied to cryptographic file systems. CTR mode properties are particularly applicable to cloud file storage, although few studies exist demonstrating their effective application in this context, or even in file systems. Important issues related to the generation, manipulation and storage of nonces are addressed. A deterministic form based on counters is presented for generation. We consider alternative forms for storage based on interleaving and separation. We evaluate the use of pread/pwrite and mmap functions for manipulation and access to these structures. In order to analyze the performance impact, we present experimental performance results by comparing a cryptographic file system operating in its traditional mode (CBC) and the implemented CTR mode.*

Resumo. *Este trabalho descreve uma abordagem inovadora de utilização do modo de operação CTR aplicado a Sistemas de Arquivo Criptográficos (SAC). As propriedades do modo CTR são particularmente aplicáveis ao armazenamento de arquivos em nuvem, embora poucos estudos existam demonstrando sua efetiva aplicação nesse contexto, ou até mesmo em sistemas de arquivos. São tratadas questões importantes relacionadas à geração, manipulação e armazenamento de nonces. Para geração é apresentada uma forma determinística baseada em contadores. Consideramos formas alternativas para armazenamento baseadas em intercalação e separação, e avaliamos o uso das funções pread/pwrite e mmap para manipulação e acesso a essas estruturas. Procurando analisar o impacto no desempenho, apresentamos resultados experimentais que comparam um SAC operando em seu modo tradicional (CBC) e o modo CTR implementado.*

1. Introdução

Uma das aplicações mais comuns da computação em nuvem é sua utilização para o armazenamento de dados. Normalmente esses serviços oferecerem canais seguros de comunicação no processo de transferência dos dados. Porém, não se preocupam em cifrar os arquivos antes de gravá-los e transferí-los, o que normalmente resulta em dados sendo armazenados na nuvem em formato de texto claro.

Uma forma de contornar o problema do armazenamento de arquivos em texto claro é utilizar um Sistema de Arquivos Criptográfico (SAC). Eles utilizam técnicas de

criptografia que permitem atender a alguns requisitos da segurança da informação como confidencialidade, integridade, autenticidade, entre outras. Atuam de forma transparente, cifrando os dados antes deles serem efetivamente armazenados. SACs podem aplicar funções de criptografia em diferentes níveis do processo de armazenamento e recuperação de dados, podendo cifrar arquivos individuais, diretórios, partições e discos inteiros. Os SACs tem demandas intensivas de processamento devido ao volume de dados e o custo das funções criptográficas. Técnicas de processamento paralelo em CPU ou GPUs podem ser usadas para atender a essas demandas. Como forma de se obter melhor desempenho nesses sistemas, é possível utilizar os recursos de processamento paralelo oferecido por computadores e servidores multiprocessados, sejam eles na forma de múltiplas CPUs ou GPUs.

SACs normalmente cifram e decifram o conteúdo dos arquivos em blocos, visando permitir acessar de forma aleatória o conteúdo dos arquivos sem a necessidade de cifrá-los ou decifrá-los completamente. Utilizam cifras de bloco e modos de operação que ditam regras específicas de como o processo da cifragem deve ser realizado. Existem vários modos de operação, entre eles o modo *Counter* (CTR). Os diferenciais do modo CTR são sua capacidade de ser totalmente paralelizável e utilizar máscaras de cifragem que podem ser geradas antes de serem utilizadas no processo final de cifragem. Tal característica pode atuar como um importante compensador em latências que são adicionadas pelo custo computacional do processo de cifragem. A utilização do modo CTR aplicada a um SAC pode trazer ganhos significativos de desempenho ao permitir aproveitar melhor os recursos de processamento paralelo, constituindo-se numa abordagem inovadora.

Neste trabalho são apresentados detalhes da implementação do modo CTR aplicado a um SAC. São tratadas questões importantes relacionadas aos *nonces* (*number used only once*), os quais são gerados a partir de um contador que é incrementado e tem papel essencial no funcionamento do modo CTR. Tal estudo tem relevância pois a falta de cuidado no tratamento dos *nonces*, além de questões de segurança, pode impactar negativamente e de forma significativa o desempenho do SAC, consequentemente anulando ganhos advindos da exploração futura das vantagens de paralelização oferecidas pelo modo.

São abordadas três questões relacionadas aos *nonces*: a primeira é uma forma determinística para sua geração, respeitando regras específicas do modo de operação CTR; a segunda trata de questões de granularidade e formas de armazenamento; a terceira estuda o desempenho de funções específicas de programação em nível de sistema para realizar sua leitura e gravação. A implementação dessas ideias foram aplicadas a um SAC já existente, denominado EncFS. Visando avaliar o impacto no desempenho do SAC, apresentamos análises de desempenho para determinar as melhores combinações das ideias implementadas. Avaliamos a aplicação desses conceitos a sistemas mais adaptados à operação em nuvem como o CryFS (seção 3), o que faz parte de nossos trabalhos em andamento.

O restante do trabalho está organizado da seguinte forma: a seção 2 apresenta os modos de operação CBC e CTR; a seção 3 apresenta trabalhos relacionados e detalha características dos SACs CryFS e EncFS; a seção 4 apresenta o sistema de entrada e saída do Linux e os níveis em que os SACs interagem com ele; a seção 5 discute ideias de implementação do modo CTR aplicado a SACs; a seção 6 apresenta uma análise do SAC EncFS onde as ideias foram implementadas; a seção 7 apresenta as conclusões.

2. Modos de Operação para Cifragem em Bloco

Por trabalhem com volumes consideráveis de dados e serem sensíveis a latência, SACs normalmente usam técnicas de criptografia simétrica por serem mais rápidas. SACs cifram os arquivos em blocos de tamanho fixo para que seja possível ter acesso a partes deles sem a necessidade de cifrá-los ou decifrá-los por completo. Os blocos são cifrados utilizando cifras de bloco que subdividem esses blocos em blocos menores (geralmente 64 ou 128 bits), processando-os individualmente [Menezes et al. 1996][Paar and Pelzl 2009][Stallings 2013].

A forma como esses blocos menores são processados durante a cifragem deve seguir regras específicas para que sejam mantidas as propriedades da segurança da informação. Essas regras constituem o que é chamado de modos de operação. A publicação especial SP 800-38A [Dworkin 2001], do *National Institute of Standards and Technology* (NIST), descreve cinco tipos básicos. Por questões de relevância e espaço, serão descritos os modos *Cipher Block Chaining* (CBC) e *Counter* (CTR).

No modo de operação CBC, cada bloco cifrado depende do bloco de texto claro (P), da chave (K) e do bloco de texto cifrado (C) imediatamente anterior. No processo de cifragem, antes de um bloco ser cifrado, ele é submetido a uma operação de `XOR` com o bloco cifrado anteriormente. A operação de cifragem pode ser vista na figura 1(a). Já o modo CTR consiste na utilização de um contador que é incrementado a cada novo bloco processado. Esse contador, denominado *nonce*, é submetido ao processo de cifragem para depois ser realizado um `XOR` com o texto a ser cifrado. A etapa final de `XOR` resulta no texto cifrado. Esse processo pode ser visto na figura 1(b).

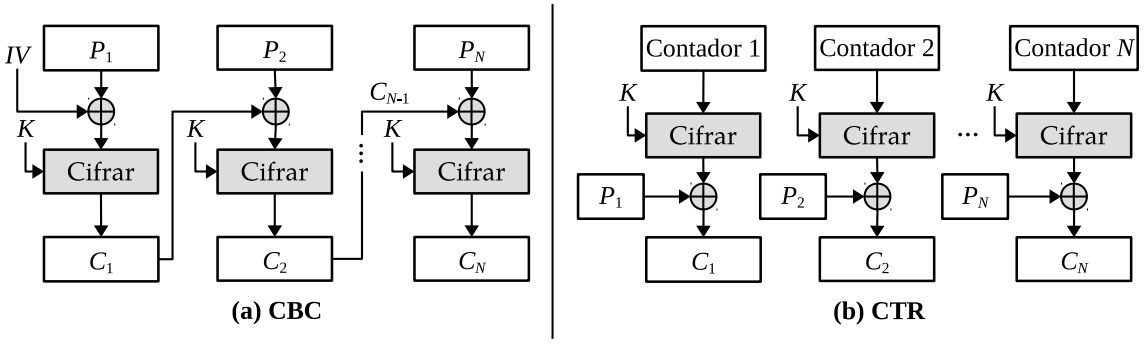

Figura 1. Modos de operação CBC e CTR (cifragem).

O modo CBC exige a utilização de um vetor de inicialização (IV). Os dados contidos no IV não podem ser previsíveis, ou seja, não pode ser possível a um atacante, com base no texto claro, prever qual será o IV utilizado num processo futuro de cifragem. A escolha dos dados a serem utilizados para popular o IV deve seguir regras específicas. Geralmente se utiliza um número gerado pseudoaleatoriamente, o qual é cifrado com a mesma chave utilizada no restante do processo de cifragem [Paar and Pelzl 2009][Stallings 2013][Ferguson et al. 2010][Dworkin 2001].

No modo CTR, a utilização do *nonce* também precisa obedecer regras específicas. A principal delas é a garantia de unicidade do par *nonce* e chave. Ou seja, não se pode utilizar mais do que uma vez o mesmo *nonce* e a mesma chave num processo de cifragem.

Isto leva a necessidade explícita de controle do incremento do *nonce* entre processos de cifragem diferentes que utilizem a mesma chave [Dworkin 2001].

Com relação à capacidade de execução paralela de cifragem dos blocos, no CBC isso não é possível. Isso se deve ao fato de que para cifrar um bloco, primeiro é preciso aguardar o término da cifragem do bloco anterior a ele. Somente pode ser paralelizado o processo de decifragem.

Em compensação, no modo CTR, por não haver dependência entre os blocos, ele é totalmente paralelizável, incluindo o processo de decifragem. Além disso, o fato da cifragem ocorrer somente sobre o *nonce* permite que ele seja feito de forma antecipada, gerando máscaras de cifragem. Assim, essas máscaras são utilizadas somente quando forem necessárias na etapa final de `XOR` com o texto claro ou cifrado.

3. Trabalhos relacionados

A utilização do modo CTR aplicado especificamente a SACs visando explorar recursos de processamento paralelo é uma abordagem pouco explorada. Boa parte dos trabalhos se concentra em questões estritamente relacionadas à aceleração em GPU das funções criptográficas. Nesse sentido, [Zola and Bona 2012] apresentam o WAES (*Warped* AES), a implementação de um algoritmo de alto desempenho para processamento paralelo heterogêneo. Nele foi implementada a cifra de bloco AES operando no modo CTR, criando uma técnica de cifragem especulativa. [Lee et al. 2016] também usam o modo CTR para acelerar o processamento em GPU dos cifradores AES, CAST, SEED, IDEA, Blowfish e Threefish. Em seus estudos exploram recursos mais recentes de GPUs NVIDIA para aprimorar o desempenho dos algoritmos.

Aplicada em SACs, a utilização de processamento em GPU das funções criptográficas também já foi explorada. No espaço do *kernel*, [Kato et al. 2012] adaptaram o SAC eCryptfs [Halcrow 2005] para utilizar um sistema de *runtime* chamado Gdev e o *driver* de vídeo Nouveau para executar as funções de criptografia em GPU. GPUStore [Sun et al. 2012] é outro exemplo de sistema de *runtime* e *framework* criado com o objetivo de facilitar a integração e tornar eficiente a utilização de processamento em GPU para sistemas que rodam no espaço do *kernel*. A efetividade de sua utilização foi comprovada ao ser aplicada aos SACs dm-crypt [Broz 2018] e eCryptfs, demonstrando ganhos significativos de desempenho.

No espaço do usuário, há outros SACs que não utilizam a aceleração das funções criptográficas em GPU, porém sua abordagem é relevante por serem bons candidatos ao processo de adaptação ao modo de operação CTR e posterior uso dos recursos de processamento em GPU. São sistemas baseados em FUSE [Rath and Szeredi 2018], o que simplifica seu desenvolvimento, configuração e utilização. Geralmente dependem de outro SA de base (SAB) onde os dados são efetivamente gravados.

O CryFS [Messmer 2018] é um exemplo de sistema baseado em FUSE. Sua arquitetura é baseada em camadas. A camada *blockstore* é responsável por ler e gravar dados de forma cifrada em blocos de tamanhos fixos. Cada bloco armazenado corresponde a um arquivo cifrado gravado no SAB. A camada *blobstore* é responsável por lidar com *blobs* de tamanhos variáveis. Ela mapeia os *blobs* para serem armazenados em um ou mais blocos. A camada *filesystem* é responsável por mapear arquivos e diretórios em *blobs*. Um arquivo ou diretório corresponde a um *blob*. A camada *fs++* faz a interface com a libfuse.

Um arquivo armazenado no CryFS não necessariamente corresponde a um arquivo armazenado no SAB. Arquivos são mapeados para *blobs*, porém de acordo com seu tamanho, eles podem ser mapeados para um ou mais blocos. Esse mapeamento e o armazenamento em blocos cifrados de tamanhos iguais garante a confidencialidade dos metadados dos arquivos e da estrutura hierárquica dos diretórios. O CryFS suporta os cifradores AES, MARS, Twofish, Serpent e CAST, disponíveis na biblioteca Crypto++. Para cifragem dos blocos ele utiliza o modo *Galois Counter Mode* (GCM).

Outro exemplo é o EncFS [Gough 2018]. Em seus processos de cifragem o EncFS pode utilizar os cifradores simétricos de bloco AES, Blowfish e Camellia, disponíveis na biblioteca OpenSSL. Para que seja possível acessar os dados de forma aleatória dentro de um arquivo cifrado, seu conteúdo é cifrado em blocos com tamanhos fixos. O tamanho é definido durante a criação do SA e pode variar de 1 até 4 KiB. Na cifragem de blocos inteiros, utiliza o modo de operação CBC. No EncFS, cada arquivo ou diretório corresponde a um arquivo ou diretório no SAB.

Na cifragem dos blocos são utilizados três tipos diferentes de IVs. O primeiro é o IV contido na chave do volume (IVV). O segundo é o IV do arquivo (IVA) e o terceiro são os IVs para cifragem dos blocos dentro dos arquivos (IVB). O IVV está armazenado nos bytes finais da chave de volume (VK), a qual é gerada aleatoriamente na criação do SA. O IVA também é gerado aleatoriamente na criação de um arquivo novo. $NumBloco$ corresponde ao número do bloco dentro do arquivo. O IVB é obtido através da aplicação de uma função de *hash* criptográfico descrita no quadro 1.

$$IVB = HMAC\_CTX(VK, IVV \parallel (NumBloco \bigoplus IVA))))$$

Quadro 1. Função de geração de IVs para cifragens de blocos.

Como o EncFS utiliza o modo CBC, os IVs utilizados na cifragem dos blocos (IVBs) precisam cumprir com o requisito de imprevisibilidade. Essa é a razão pela qual é feita a concatenação de diferentes IVs e a aplicação da função de *hash* criptográfico. Considerando um mesmo arquivo armazenado no mesmo SA EncFS, VK, IVV e IVA mantém-se constantes, sendo a única variável o número do bloco. Essa característica traz vantagens com relação ao desempenho, pois permite que os IVBs sejam calculados dinamicamente, não sendo necessário armazená-los. Além disso, o mesmo IVB pode ser reutilizado em processos de regravação de um mesmo bloco dentro de um mesmo arquivo. O IVA possui 8 bytes e é armazenado no cabeçalho do arquivo.

Os trabalhos citados no começo da seção demonstram a efetividade do processamento em GPU para as funções criptográficas, inclusive ao serem aplicadas a SACs. Contudo, considerando os SACs que realizam o processamento em GPU, nenhum deles faz uso do modo CTR e consequentemente de seus benefícios já apresentados. Os SACs CryFS e EncFS podem ser aplicados facilmente no contexto do armazenamento de arquivos na nuvem, porém não utilizam o processamento em GPU. Este trabalho marca o início do processo que visa contornar essa limitação, ao preparar um desses SACs (EncFS) para operar no modo CTR, permitindo, posteriormente, que o mesmo execute as funções criptográficas em GPU.

4. Sistema de entrada e saída e interação com SACs

A função de um sistema operacional (SO) é oferecer uma camada de abstração clara e simplificada que permita às aplicações terem acesso a recursos de hardware como memória, processadores, dispositivos de armazenamento de dados, entre outros. Também se encarrega de gerenciar e controlar o acesso a esses recursos. No SO Linux essas funções estão implementadas em três sistemas principais: controle de entrada e saída (E/S), gerenciamento de memória e gerenciamento de processos. Eles formam o que é chamado de *kernel* do SO, estando situados em um espaço denominado espaço do *kernel* [Tanenbaum and Bos 2014][Love 2010].

Os subcomponentes que fazem parte do sistema de controle de E/S estão organizados nas seguintes camadas: *driver* de dispositivo, camada genérica de blocos/escalonador, sistema de arquivos e *Virtual File System* (VFS). A figura 2 ilustra sua organização.

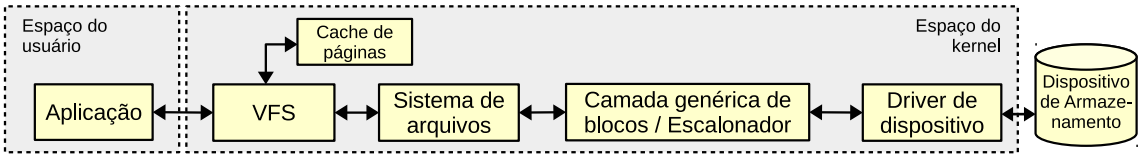

Figura 2. Subcomponentes do sistema de E/S do *kernel* Linux.

Através do *driver* de dispositivo tem-se acesso e controle sobre dispositivos de armazenamento onde os dados são gravados. Antes das requisições de leitura e gravação chegarem ao *driver* de dispositivo, elas passam pela camada genérica de blocos, sofrendo a ação do escalonador de requisições, o qual procura ordenar e fundir as requisições visando otimizar o acesso físico aos dispositivos.

A camada de sistema de arquivos (SA) é a intermediária entre o VFS e a camada genérica de blocos. SAs que funcionam no espaço do *kernel* situam-se nela e possuem formas distintas de funcionamento. É nessa camada que estão implementas as várias funções associadas a arquivos e diretórios, as quais são chamadas através do VFS. Por sua vez, o VFS funciona como uma camada de abstração onde estão definidas interfaces e estruturas de dados que são suportados pelos diferentes SAs. O sistema de E/S também inclui o mecanismo de *cache* de páginas, o qual utiliza a memória de acesso aleatório (RAM) para realizar o *cache* de dados que são lidos e gravados através do VFS. Por fim, têm-se as aplicações em nível de usuário.

De acordo com sua arquitetura, os sistemas de armazenamento criptográficos podem interagir em diferentes níveis do sistema de E/S. Em se tratando do nível de confidencialidade dos dados, os sistemas que operam junto à camada genérica de blocos são mais abrangentes. Ao permitir cifrar partições e discos inteiros, escondem informações como metadados de arquivos e estrutura hierárquica de diretórios. Além disso, permitem manter a confidencialidade de dados armazenados em partições de *swap* e temporárias.

Geralmente SACs que atuam no espaço do usuário possuem maior flexibilidade, mas com desempenho menor devido às constantes necessidades de trocas de contexto entre o espaço do usuário e do *kernel*. Porém, oferecem maior flexibilidade de uso, pois não exigem privilégios elevados de usuário nos processos de desenvolvimento, configuração e utilização do SA. A figura 3 ilustra como alguns dos SACs anteriormente citados se

situam nas diferentes camadas do sistema de E/S. A camada genérica de blocos está representada como E/S de blocos e a camada de *driver* de dispositivo foi omitida.

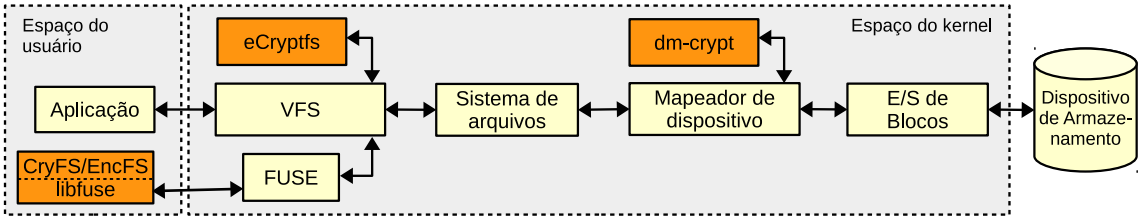

Figura 3. Posicionamento dos SACs no sistema de E/S.

A seção seguinte descreve a implementação do modo CTR e sua aplicação a um SAC que roda no espaço do usuário.

5. Implementação do modo CTR aplicado a um SAC

Implementar o modo CTR em um SAC traz a vantagem de poder explorar recursos de processamento paralelo de computadores multiprocessados. Além do ganho no desempenho advindo da execução paralela das funções criptográficas, a possibilidade de computação antecipada das máscaras de cifragem também pode representar uma contribuição significativa no ganho de desempenho ou atuar como um importante compensador em latências adicionais advindas do processo de paralelização.

Contudo, realizar uma implementação ingênua do modo CTR pode fazer com que haja queda significativa no desempenho do SAC, reduzindo ou até mesmo anulando possíveis ganhos advindos do processamento paralelo. Por isso, é importante que algumas questões sejam observadas, principalmente as relacionadas com a geração, armazenamento e acesso aos *nonces*.

Atendendo ao requisito de unicidade exigido pelo modo CTR, é preciso que a geração de *nonces* seja feita de forma controlada. Aplicado a um SAC, isso significa que nos processos de gravação e regravação de blocos, é necessário que haja um mecanismo de controle, sendo responsável por gerar *nonces* únicos para cada operação de gravação e regravação de blocos.

Operações de leitura e gravação sobre arquivos armazenados em um SAC envolvem o processamento de vários blocos. O mesmo *nonce* utilizado na cifragem de um bloco precisa ser utilizado no processo inverso. Portanto, é necessário que ele seja armazenado para posterior recuperação. A granularidade com que os *nonces* são trabalhados e a forma com que são armazenados podem afetar significativamente o desempenho do SAC. Além disso, o acesso aos *nonces* ocorre com alta frequência. Há várias funções disponíveis em C/C++ que podem ser utilizadas para controlar a leitura e gravação dos *nonces*. É importante analisar quais delas oferecem um melhor desempenho.

As subseções seguintes apresentam diferentes ideias que procuram tratar essas questões, justamente no contexto da implementação do modo CTR aplicado a um SAC.

5.1. Geração dos *nonces*

Para atender ao requisito de unicidade do *nonce* exigido pelo modo CTR, foi implementada uma forma determinística para sua geração [Dworkin 2001]. A ideia é baseada em contadores que são incrementados de forma controlada.

Na utilização de uma cifra de bloco de 128 bits, o *nonce* também precisa ter 128 bits. Esses 128 bits podem ser utilizados para armazenar dois contadores de 64 bits. O contador armazenado nos primeiros 64 bits é o contador do SA (CSA), o segundo é o contador do arquivo (CA). O CSA é incrementado sempre que um novo arquivo é criado ou um arquivo solicita um novo contador. O CA é gerenciado pelo próprio arquivo e incrementado a cada nova gravação e regravação de bloco. O formato dos contadores, que acabam sendo utilizados como parâmetro de entrada (*nonce*) das funções de cifragem do modo CTR da biblioteca OpenSSL, podem ser vistos na figura 4.

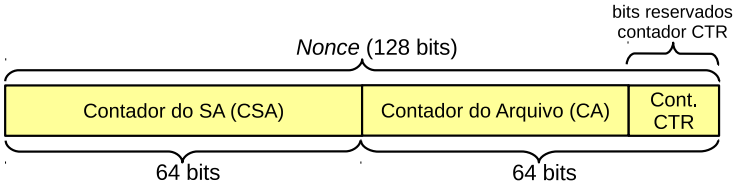

Figura 4. Contadores utilizados na formação de um *nonce*.

De acordo com o tamanho do bloco, definido na criação do SAC, são reservados $log_2(x/16)$ bits menos significativos do segundo contador, onde x corresponde ao tamanho do bloco em bytes. Esses bits são utilizados para armazenar o contador que será incrementado no funcionamento interno da implementação do modo CTR da biblioteca OpenSSL. Os bis menos significativos são preservados incrementando-se o CA através da fórmula $x = x + (1 << y)$, onde x corresponde ao valor atual do contador e y a quantidade de bits reservados. A quantidade total de blocos que se pode ler ou gravar no SAC é dada pelo valor 2^{128-x}, onde x correspondente à quantidade de bits reservados.

Os contadores CSA e CA podem iniciar com valores aleatórios. Tanto seus valores iniciais quanto atuais são armazenados. O valor atual pode ser incrementado até que ocorra *overflow* e depois do valor zero até o valor do contador inicial menos um. Quando o CA se esgota, pode-se gerar um novo CSA e CA, de tal forma que seja possível continuar gravando e regravando blocos de um arquivo. Caso o CSA também se esgote, esse se recusa a gerar novos contadores, não sendo mais possível gravar e regravar blocos para novos arquivos ou arquivos que já possuam seu CA esgotado. Esse controle garante que o requisito de unicidade do *nonce* não seja violado. O armazenamento do CSA e dos valores inicial e atual do CA é feito no cabeçalho dos arquivos.

5.2. Armazenamento dos *nonces*

Quando um bloco é gravado, o *nonce* utilizado no seu processo de cifragem é formado pela junção do CSA e do CA. Num processo futuro de leitura do mesmo bloco, para que seja possível decifrar seu conteúdo, é necessário que o mesmo *nonce* seja passado como parâmetro às funções de decifragem. Portanto, é necessário que ele seja armazenado para futura recuperação.

Uma ideia simples seria armazenar o *nonce* individualmente antes de cada bloco. Porém, isto levaria a um aumento significativo da latência, principalmente nos processos de leitura aleatória, devido à necessidade constante e em grande quantidade da utilização de operações de *seek* para realizar a leitura prévia e individual de cada *nonce*. Além disso, como os *nonces* possuem 128 bits, isso resultaria numa estrutura de armazenamento não

alinhada com o tamanho comum de páginas de memória do SO e blocos de dados dos SAs, prejudicando seu desempenho.

Foram implementadas e avaliadas duas formas de armazenamento dos *nonces*. A primeira foi armazenando-os em grupos. Cada grupo de *nonce* possui um tamanho fixo, podendo armazenar uma quantidade específica de *nonces*. Os grupos de *nonces* são armazenados de forma intercalada dentro do arquivo. Cada grupo de *nonce* precede uma quantidade específica de blocos de dados a que se referem. A figura 5(a) ilustra o formato do arquivo onde são utilizados grupos de *nonces* com tamanho de 4 KiB.

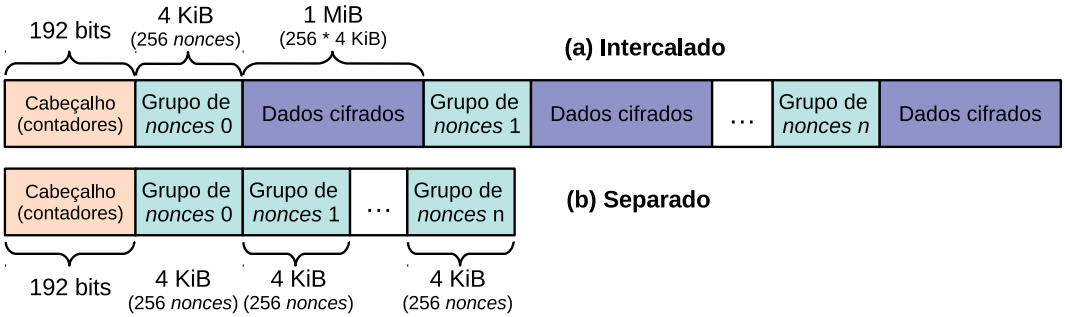

Figura 5. As duas formas de armazenamento dos grupos de *nonces*.

A segunda forma visa permitir que os dados referentes aos *nonces* possam ser armazenados de maneira contígua em disco, agilizando operações de leitura e gravação aleatórias. Seu armazenamento é feito em arquivos separados dos arquivos que contém os dados cifrados. Dessa forma, cada arquivo armazenado no EncFS tem outro arquivo contendo somente o cabeçalho com os contadores e os grupos de *nonces*. Essa forma de armazenamento pode ser vista na figura 5(b).

5.3. Funções para controle de leitura e gravação dos *nonces*

Foram escolhidas as funções pread/pwrite e mmap para realizar a leitura e gravação de *nonces*. Pread/pwrite são funções simples e, teoricamente, de baixo custo. A grande vantagem de seu uso reside no fato de se poder ter um controle maior de quando os dados são efetivamente lidos ou gravados em disco. Assim, é possível ler grupos de *nonces* do disco e mantê-los totalmente em memória, agilizando seu acesso e processamento. Somente quando for mais apropriado esses grupos são gravados novamente em disco.

A desvantagem do controle explícito de leitura e gravação em disco eleva o risco de se perder a integridade dos dados. Caso haja grupos de *nonces* carregados em memória, contendo *nonces* modificados, e ocorra algum problema na aplicação, SO, etc, os blocos de dados cifrados correspondentes a esses *nonces* se tornam impossíveis de serem decifrados. É necessário haver um cuidado especial na sincronia dos dados entre memória e disco. Na implementação, essa sincronia é realizada nas operações de fechamento do arquivo e sempre que uma requisição de sincronia (*sync*) é realizada sobre um arquivo.

A segunda opção é a função mmap. Ela permite mapear regiões de dados armazenados em disco em páginas de memória. É o SO quem se encarrega de fazer a leitura dos dados em disco e cópia dos mesmos para os endereços mapeados, bem como, no caso da gravação, pegar as páginas de memória mapeadas e marcadas como sujas e gravá-las

novamente em disco. Mapear grupos de *nonces* armazenados em disco em memória simplifica seu manuseio e ajuda a reduzir os riscos de perda de integridade do SA. Ao se alterar um *nonce* de um grupo de *nonces* mapeados, as páginas de memória que armazenam esse mapeamento são marcadas como sujas, sendo as mesmas automaticamente gravadas em disco na próxima operação de escrita de páginas sujas feitas pelo SO.

Uma desvantagem da utilização do mmap, principalmente nesse cenário aplicado ao controle dos *nonces*, é que ele só pode ser usado para mapear endereços com *offset* que sejam múltiplos do tamanho das páginas de memória. Nos casos de controle dos *nonces* em processos de gravação, onde não se sabe o tamanho final que o arquivo sendo armazenado terá, torna obrigatório que o aumento do tamanho dos arquivos de *nonces* seja incrementado em múltiplos desse valor. Isto pode levar a um desperdício de espaço em disco, principalmente num cenário com vários arquivos menores do que 4 KiB.

As ideias anteriormente descritas visando cumprir com o requisito de unicidade dos *nonces*, seu armazenamento e uma forma eficiente de acesso a eles, levou a necessidade de se mensurar o desempenho de sua implementação. Para evitar o retrabalho de criar um novo SAC, optou-se por implementar o modo CTR aplicando-o a um SAC já existente. Foi escolhido o EncFS por ser um SAC conhecido e baseado em FUSE, o que ajuda a simplificar e agilizar os processos de desenvolvimento e testes, além de poder ser utilizado em conjunto com serviços de armazenamento na nuvem.

6. Resultados e discussão

Para se definir qual a melhor forma de armazenamento dos *nonces* (em arquivos separados ou intercalados), bem como qual função de leitura e gravação apresenta melhor desempenho (pread/pwrite ou mmap), foram feitas medições de vazão utilizando-se a ferramenta `fio` versão 3.2-72, configurada para utilizar um único arquivo com 16 GiB (dobro da memória RAM). Foi utilizada a opção `refill_buffers` para que os dados gravados no arquivo fossem totalmente aleatórios e a opção `allrandrepeat` para que todas as funções baseadas em aleatoriedade apresentassem o mesmo comportamento entre diferentes execuções da ferramenta. A ferramenta foi configurada para enviar uma requisição de sincronia a cada 16 MiB de dados gravados.

Foram medidos níveis de vazão com operações variando o tamanho das requisições de 1 KiB até 512 KiB. As requisições vindas da libfuse por padrão estão limitadas a 32 páginas de memória (128 KiB), portanto não há a necessidade de medir requisições com tamanhos significativamente maiores do que 128 KiB. Foram medidos quatro tipos diferentes de operação: escrita sequencial, escrita aleatória, leitura sequencial e leitura aleatória. Cada medição foi executada por um período de 60 segundos e repetida 10 vezes, sendo calculada a média aritmética simples dos resultados. Entre cada repetição foi executado um comando para realizar o descarte das páginas de memória em *cache*. Foi utilizada uma partição específica para a criação do SAB. Entre os diferentes cenários avaliados, o SAB era desmontado, recriado e remontado. O SAB utilizado foi o ext4 com as configurações padrões.

Os testes foram realizados utilizando-se o Linux com *kernel* versão 4.4.0, rodando em processador Intel Core i7 920 a 2,67 GHz (frequência fixa), 8 GiB de memória RAM DDR3 atuando em *dual channel* e disco Western Digital modelo WD5000AZLX-0 com taxa de vazão sustentada teórica máxima por volta de 146.484 KiB/s. A versão utilizada

da biblioteca libfuse foi a 2.9.2 e da biblioteca OpenSSL 1.0.1f.

O cuidado na preparação do ambiente de testes teve por objetivo criar um ambiente que fosse o mais homogêneo possível. Os principais objetivos foram: (i) Procurar reduzir ao máximo a atuação do mecanismo de *cache* de páginas, entretanto, sem anulá-lo; (ii) Ler e gravar dados aleatórios reais e com os mesmos padrões de acesso aleatório; (iii) Gravar e ler dados sempre nas mesmas regiões em disco.

As figuras 6, 7, 8 e 9 apresentam as medições de vazão para as quatro combinações possíveis. Para efeito de comparação também foi incluída a vazão do EncFS operando em seu modo padrão de operação, o CBC. No eixo secundário y é apresentado o ganho ou perda de vazão comparando as diferentes combinações e o modo CBC.

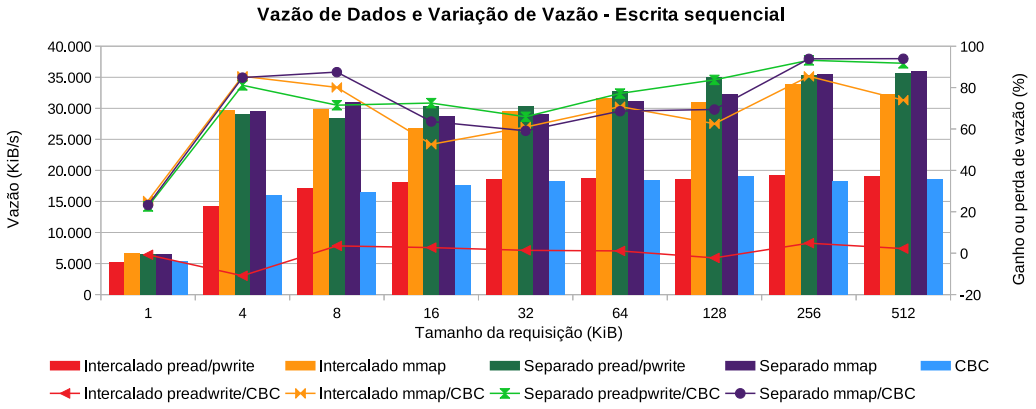

Figura 6. Vazão e variação de vazão em escrita sequencial.

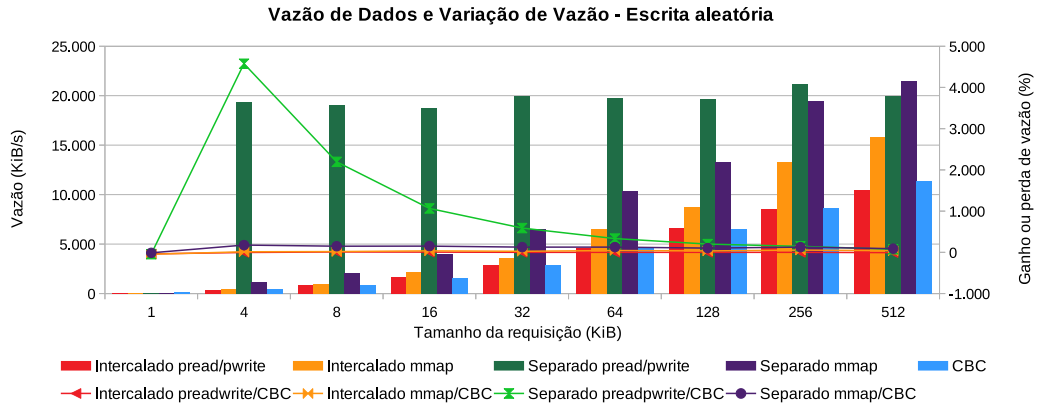

Figura 7. Vazão e variação de vazão em escrita aleatória.

Na escrita sequencial, a combinação com desempenho claramente inferior é a de *nonces* armazenados intercaladamente com acesso via pread/pwrite. Considerando-se a média dos valores obtidos em cada tamanho de requisição, não há ganho se comparado ao modo CBC e uma perda de 10% no caso de requisições de 4 KiB. As outras três combinações apresentam desempenho semelhante. Grupos de *nonces* intercalados acessados via mmap apresentam desempenho médio 66% superior ao CBC. *Nonces* separados com acesso via pread/pwrite 73% e com mmap 71% superiores ao CBC.

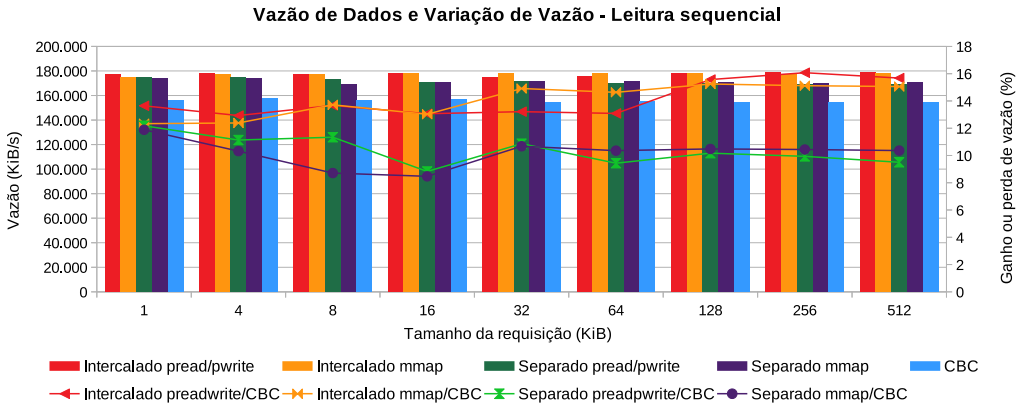

Figura 8. Vazão e variação de vazão em leitura sequencial.

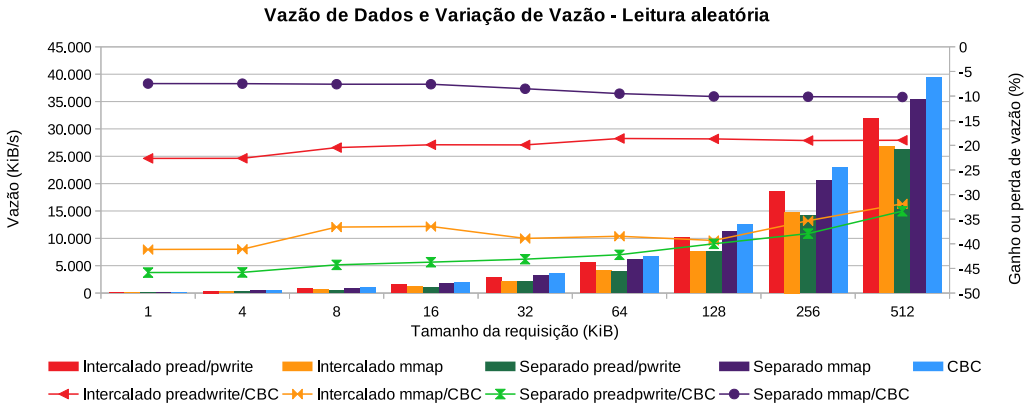

Figura 9. Vazão e variação de vazão em leitura aleatória.

Na escrita aleatória, *nonces* intercalados com acesso via pread/pwrite apresentam perda média de 4%. Intercalados com acesso via mmap apresentam ganho médio de 23%. Separados com mmap, ganho médio de 114%. O destaque nesse cenário é da combinação de *nonces* separados com acesso via pread/pwrite. Nesse caso, desde as requisições com 4 KiB, consegue-se atingir o nível de vazão máximo, situado próximo da casa dos 20.000 KiB/s. O ganho médio chega a ser de 1014%.

O cenário da leitura sequencial não traz grandes diferenças entre as quatro combinações. Claramente nesse caso o gargalo na vazão é ocasionado pela taxa de vazão suportada pelo disco rígido, o que não permite analisar níveis maiores de diferença entre as implementações. *Nonces* intercalados tem ligeira vantagem em relação aos *nonces* separados. Em média seu desempenho é superior 14% se comparado ao CBC. *Nonces* separados apresentam desempenho em média 10% superior.

Conforme previsto, o cenário onde as perdas de desempenho se manifestam é na leitura aleatória. *Nonces* separados com acesso via pread/pwrite tem as maiores perdas, em média 41%, seguido pelos *nonces* intercalados com acesso via mmap, com perda média de 37% e intercalados com acesso via pread/pwrite com 20%. A combinação que apresenta o menor valor de perda é dos *nonces* separados com acesso via mmap, com perda média situada na casa dos 8%.

Considerando a média geral dos quatro cenários, o melhor desempenho é da combinação de *nonces* separados com acesso via pread/pwrite, onde o ganho médio foi de 246%, seguido pela combinação de *nonces* separados com acesso via mmap com ganho médio de 46% e *nonces* intercalados com acesso via mmap com ganho médio de 16%. A única combinação cujo desempenho geral médio apresentou perda foi a de *nonces* intercalados com acesso via pread/pwrite, com perda média de 2%.

É importante ressaltar que apesar da combinação de *nonces* separados com acesso via pread/pwrite ter a maior média geral, não se pode desconsiderar a perda significativa na leitura aleatória de 41%. Por isso, mesmo ficando em segundo lugar na média geral, a combinação de *nonces* separados com acesso via mmap, onde a perda na leitura aleatória é de apenas 8%, parece ser a mais equilibrada entre as quatro combinações mensuradas.

7. Conclusões e trabalhos futuros

Este trabalho explorou a implementação do modo de operação CTR aplicado a um SAC. Manteve-se o foco sobre um elemento importante do modo CTR, chamado *nonce*. Foram trabalhadas questões relacionadas a sua geração, armazenamento e manipulação. Foi apresentada uma forma determinística para sua geração, visando cumprir com a regra de unicidade exigida pelo modo. Também foram exploradas duas formas diferentes de armazená-los, agrupando-os e intercalando-os dentro dos arquivos cifrados e exclusivamente em arquivos separados. Por último, foram estudadas funções específicas da linguaguem C/C++, pread/pwrite e mmap, para realizar a leitura e gravação dos grupos de *nonces*.

A utilização de duplos contadores para geração dos *nonces* se mostrou apropriada, garantindo sua unicidade e não prejudicando o desempenho do SAC. Com relação ao armazenamento dos *nonces* e sua manipulação, foram feitas análises de desempenho medindo a vazão com quatro combinações diferentes e comparadas com o desempenho normal do SAC operando no modo padrão CBC. A combinação com armazenamento de grupos de *nonces* em arquivos separados, sendo acessados através da função mmap, apresentou ganho de desempenho mais equilibrado. Considerando-se a média de todas as operações, seu desempenho foi 46% superior ao modo CBC. Todas as combinações apresentaram perdas na leitura aleatória, porém, nesse último caso, a perda foi em média apenas de 8%.

Como trabalhos futuros, tem-se o aprimoramento do armazenamento e controle dos arquivos de *nonces* para tentar contornar o problema de desperdício no uso de espaço em disco para arquivos pequenos, além de melhorar o desempenho na leitura aleatória. Pretende-se implementar um esquema de armazenamento inspirado nos *inodes* do Unix. Além disso, com a implementação do modo de operação CTR feita neste trabalho, preparou-se o caminho para se explorar os recursos de processamento paralelo das funções de criptografia. Isto pode ser muito promissor no sentido de permitir utilizar de forma mais efetiva ambientes multiprocessados com CPUs e GPUs, melhorando significativamente o desempenho dos SACs.

Pretende-se aplicar essas ideias também ao SAC CryFS, cuja construção já foi pensada em garantir a segurança de dados armazenados na nuvem, inclusive com recursos que garantem a confidencialidade de metadados dos arquivos e estrutura hierárquica de diretórios, além de sua integridade. Apesar de operar no modo GCM, o modo CTR possui similaridades, o que permite aplicar as mesmas ideias desenvolvidas neste trabalho.

Referências

Broz, M. (2018). dm-crypt: Linux Kernel Device-Mapper Crypto Target. `https://gitlab.com/cryptsetup/cryptsetup/wikis/DMCrypt`. [Online; accessed 15-February-2018].

Dworkin, M. J. (2001). SP 800-38A 2001 Edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Technical report, Gaithersburg, MD, United States.

Ferguson, N., Schneier, B., and Kohno, T. (2010). *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing.

Gough, V. (2018). EncFS: An Encrypted Filesystem for FUSE. `https://github.com/vgough/encfs`. [Online; accessed 15-February-2018].

Halcrow, M. A. (2005). eCryptfs: An Enterprise-class Cryptographic Filesystem for Linux. In *Proceedings of the 2005 Linux Symposium*, volume 1, pages 201–218.

Kato, S., McThrow, M., Maltzahn, C., and Brandt, S. (2012). Gdev: First-Class GPU Resource Management in the Operating System. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 37–37, Berkeley, CA, USA. USENIX Association.

Lee, W.-K., Cheong, H.-S., Phan, R. C.-W., and Goi, B.-M. (2016). Fast Implementation of Block Ciphers and PRNGs in Maxwell GPU Architecture. *Cluster Computing*, 19(1):335–347.

Love, R. (2010). *Linux Kernel Development*. Addison-Wesley Professional, 3rd edition.

Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. (1996). *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

Messmer, S. (2018). CryFS: A Criptographic Filesystem for the Cloud. `https://www.cryfs.org`. [Online; accessed 15-February-2018].

Paar, C. and Pelzl, J. (2009). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Publishing Company, Incorporated, 1st edition.

Rath, N. and Szeredi, M. (2018). The Reference Implementation of the Linux FUSE Interface (libfuse). `https://github.com/libfuse/libfuse`. [Online; accessed 15-February-2018].

Stallings, W. (2013). *Cryptography and Network Security: Principles and Practice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 6th edition.

Sun, W., Ricci, R., and Curry, M. L. (2012). GPUStore: Harnessing GPU Computing for Storage Systems in the OS Kernel. In *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR '12, pages 9:1–9:12, New York, NY, USA. ACM.

Tanenbaum, A. S. and Bos, H. (2014). *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition.

Zola, W. M. N. and Bona, L. C. E. D. (2012). Parallel Speculative Encryption of Multiple AES Contexts on GPUs. In *2012 Innovative Parallel Computing (InPar)*, pages 1–9.