# UM MODELO DE INDEXAÇÃO E RECUPERAÇÃO PRIVADA DE DOCUMENTOS

**Washington Fonseca Santos[1], Carlos Henrique Figueiredo Felipe[1], Talles Brito Viana[1]**

[1] Instituto Federal de Educação, Ciência e Tecnologia do Ceará – *campus* Crato

*{washingtonstk, cahe7cb}@gmail.com, tallesbrito@ifce.edu.br*

***RESUMO:*** *À medida que sistemas de computação se tornam cada vez mais integrados, tem-se que a quantidade de dados que é coletada e armazenada por esses sistemas também aumenta. Sistemas de computação modernos necessitam ser capazes de lidar com grande quantidade de dados ao mesmo tempo em que devem oferecer mecanismos apropriados para a segurança desses dados. Neste cenário, prover a privacidade é um desafio. Este trabalho apresenta um modelo de indexação e recuperação privada de dados, tal que nem mesmo o servidor que armazena os índices e realiza as consultas conhece o conteúdo dos arquivos ou das consultas.*

***Palavras-chave:*** *Indexação. Recuperação de Informação. Privacidade.*

***ABSTRCT:*** *As computing systems become more integrated, the amount of data that is collected and stored by those systems has grown. Modern computing systems must be capable of handling such large amount of data and besides should provide suitable mechanisms for securing this data. In such a scenario, providing data privacy is a current challenge. This work presents a private document indexing and information retrieval model in which the server stores indexes and runs queries without knowing the document indexed content.*

***Keywords:*** *Indexing. Information Retrieval. Privacy.*

## 1. INTRODUÇÃO

Com o aumento do uso de recursos computacionais uma grande quantidade de dados, dados que podem ser públicos ou extremamente privados são criados a todo o momento. Além disso, em função da popularização dos dispositivos móveis como *tablets*, *smartphones*, *smartwatches* e outros dispositivos, em muitas situações é viável tornar os dados destes dispositivos acessíveis na rede através de um servidor local ou remoto.

Atualmente, é comum para instituições bem como para usuários individuais terceirizar o armazenamento de suas informações para serviços especializados (tais como *Amazon AWS*, *Google Cloud Platform*, *Dropbox*, *MEGA*), já que assim, é possível obter maior disponibilidade dos dados, e também, redução de custos, se comparado com o caso em que a instituição ou usuário opte por implantar a estrutura necessária para armazenar as informações. Além disso, alcança-se a isenção de responsabilidades, uma vez que o serviço contratado passa a carregar a obrigação de manter os dados acessíveis sempre que necessário (VIEIRA, 2017).

Com a popularização destas soluções, cresce também o risco da utilização das mesmas, principalmente quando se armazenam dados sensíveis, tais como documentos pessoais, senhas e mensagens. Estes serviços garantem o controle de acesso aos dados em relação ao acesso não autorizado de terceiros, apesar disso, os provedores do serviço podem coletar dados e armazená-los para direcionamento de anúncios, aperfeiçoamento de serviços e outros propósitos. Para evitar essa utilização é comum que seja realizada a encriptação dos arquivos antes de armazená-los, já que uma vez encriptados, estes apenas poderão ser lidos por quem possuir a chave para desencriptação, restringindo assim o acesso ao conteúdo dos mesmos.

Depois de armazenar os arquivos, em algum momento, será necessário utilizá-los, o que é simples quando se tem arquivos armazenados em texto plano. Em contrapartida, essa tarefa se torna mais complexa quando os mesmos estão encriptados. O conceito de manter dados sensíveis a salvo de acessos não autorizados é crucial hoje, mesmo quando os dados são armazenados por terceiros. Isto se torna importante visto que as instituições estão se tornando cada vez mais digitais e distribuídas por várias localizações físicas. Até mesmo para usuários individuais é importante manter a privacidade dos dados através da encriptação e restrição de acesso juntamente com a facilidade proporcionada pelo armazenamento terceirizado.

Dado esta problemática, este trabalho apresenta um modelo de indexação e recuperação de informação que busca garantir a privacidade dos arquivos armazenados pelo usuário, em que apenas usuários autorizados conseguem ter acesso aos arquivos (os quais podem ser documentos com conteúdo textual) e realizar consultas textuais sob os mesmos.

Para isso, na Seção 2 são discutidos tópicos

pertinentes ao tema de recuperação de informação. Na Seção 3, o modelo de indexação e recuperação privada de documentos definido neste trabalho é descrito. Na Seção 4 questões sobre a implementação do modelo discutido neste trabalho são apresentadas. Na Seção 5, um exemplo de uso de uma ferramenta que emprega o modelo definido neste trabalho é ilustrado. Já na Seção 6 é discutida uma avaliação experimental do modelo definido. E por fim, a Seção 7 é destinada a considerações finais e trabalhos futuros.

## 2. FUNDAMENTAÇÃO TEÓRICA

Diante da realidade atual, é um problema em aberto manter dados acessíveis em diversos dispositivos e em várias localidades físicas mantendo todos esses dados seguros tanto em função do acesso não autorizado por terceiros, quanto na confidencialidade dos mesmos para quem os armazena. Têm sido estudadas diferentes maneiras de prover uma solução, dentre estas o uso de *clouds* encriptados com tecnologias de encriptação de blocos, sincronização de dados e transporte seguro de informações, tais como o EncFS (ENCFS, 2017), CryFS (MESSMER, 2017).

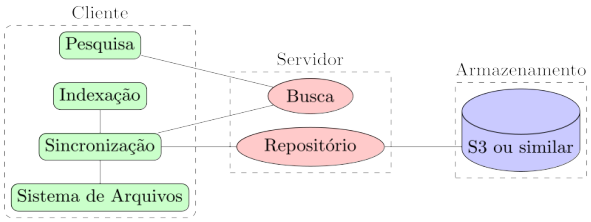
Sem dúvidas, localizar informações quando se tem um alto número de dados concentrados em um local pode se tornar uma tarefa árdua. Visando amenizar esta situação, pode-se contar com o apoio da Recuperação de Informação (*IR - Information Retrieval*), que segundo Manning, Raghavan e Schütze (2008), pode ser definida como um conjunto de técnicas para encontrar materiais que satisfaçam uma determinada necessidade de informação. Tais materiais comumente estão armazenados em computadores. Contudo, técnicas de IR podem apresentar problemas de segurança, uma vez que, podem permitir a reconstrução de um documento a partir do índice e também a obtenção do conteúdo das consultas realizadas. Com a intenção de solucionar tais vulnerabilidades, tem sido definida a Recuperação Privada de Informações (*PIR – Private Information Retrieval*), esta que por sua vez faz uso das técnicas de IR e também de esquemas de encriptação e *hashing* para garantir que tanto os índices gerados quanto o conteúdo das consultas estarão seguros, de forma que apenas o proprietário dos documentos possa ter acesso a tais conteúdos.

Dado este contexto, o modelo de recuperação e informação discutido neste trabalho visa permitir a indexação e recuperação privada de documentos de maneira que somente o usuário local possa ter acesso às informações dos documentos indexados mesmo que estes estejam armazenados em serviços de armazenamento remoto (*cloud*).

## 3. MODELO DEFINIDO

O modelo de indexação e recuperação privada de documentos definido neste trabalho é composto por um conjunto de componentes sendo executados no computador do usuário (aqui denominado de *cliente*) e outro conjunto de componentes sendo executados em um ou mais servidores remotos (aqui denominado de *servidor*). Vale ressaltar que é possível também adicionar serviços especializados em armazenamento de dados como o *Amazon S3*. Conforme ilustrado na Figura 1, no cliente existem quatro (4) principais módulos que são: *Pesquisa*; *Indexação*; *Sincronização*; e o *Sistema de Arquivos* (cada um destes serviços será detalhado a seguir neste trabalho). Enquanto isso, no servidor existem dois (2) serviços que são: *Motor de Busca*; e *Repositório*.

**Figura 1. Representação do modelo definido.**




Da perspectiva do cliente, tem-se que o módulo de Pesquisa é responsável por receber uma consulta do usuário, prepará-la para ser enviada ao motor de busca do servidor e lidar com os resultados retornados pelo motor de busca. Já o módulo de Indexação recebe eventos do sistema de arquivos para assim realizar a indexação de um documento sempre que necessário (o processo de indexação é detalhado na Seção 3.1). O módulo de Sincronização é responsável por manter o repositório local e o remoto sempre atualizados, bem como, realizar o envio dos índices seguros gerados pelo serviço de indexação. Por fim, o Sistema de Arquivos é o local em que a rotina de encriptação de blocos é executada de maneira transparente ao usuário.

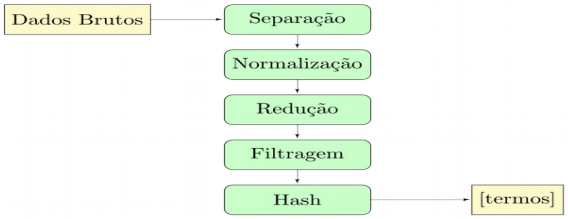
No servidor, tem-se que o Motor de Busca é responsável por receber consultas do módulo de pesquisa do cliente, realizar o processamento das consultas e retornar os possíveis resultados. O módulo de Repositório tem por finalidade autenticar os usuários e agrupar os blocos encriptados recebidos, atuando diretamente com o módulo de sincronização do cliente. Vale ressaltar que na perspectiva do servidor é possível adicionar uma nova camada de armazenamento externo, como o Amazon S3 ou serviços de *cloud* semelhantes.

### 3.1. Indexação

O módulo de Indexação recebe do sistema de arquivos um sinal de que um arquivo foi fechado e então verifica se o arquivo contém informações textuais que possam ser indexadas. Caso haja informações textuais, o arquivo

passa pelo processo de extração: esse processo consiste em extrair de um arquivo apenas o conteúdo textual, descartando qualquer tipo de formatação e outros elementos visuais não textuais. Sendo assim, o resultado do processo de extração é a obtenção de uma cadeia de caracteres (uma *string*) que contém todos os elementos textuais do arquivo. Em seguida esses dados textuais extraídos diretamente do arquivo (os quais neste trabalho são denominados de *dados brutos*) passam por algumas verificações e alterações em seu conteúdo, conforme ilustrado na Figura 2.

**Figura 2. Etapas que ocorrem na indexação de um arquivo.**




Primeiramente ocorre a etapa de *Separação*, que consiste em verificar quais dos elementos textuais presentes nos dados brutos são palavras e em seguida retorna-se uma lista com o resultado dessa separação. Na sequência, as verificações realizadas envolvem o processo de *Normalização*, que realiza o ajuste de caso dos termos, de forma que todos sejam escritos em letras minúsculas. Após isso, é realizada a *Redução* do conjunto, que consiste em remover termos duplicados, permitindo mais eficiência tanto na indexação quanto no armazenamento dos índices e na busca, essa etapa por sua vez retorna um novo conjunto de termos únicos. Em seguida, esses termos únicos passam por uma *Filtragem* de *stop words*, que é o conjunto de palavras comuns a um idioma e que são descartadas pelo mecanismo de busca, pois não agregam valor à consulta e o resultado dessa filtragem é o conjunto de termos que representa o arquivo. Após obter um conjunto de termos normalizados, reduzidos e filtrados $T$ tal que $T_i \neq T_j$ com $i \neq j$, é gerado um novo conjunto $H$ a partir do cálculo de *hash* de cada termo $T_i$ com a chave do usuário. Assim caso dois usuários diferentes manipulem o mesmo arquivo, o conjunto resultante após aplicação da função de *hashing* é diferente. Vale ressaltar que para cada arquivo indexado é gerado um novo conjunto $H$ de termos indexados.

A estrutura de dados que é empregada na indexação é o Filtro de Bloom (BLOOM, 1970) (Bonomi et al., 2006), o qual é uma estrutura de dados probabilística e eficiente para o armazenamento. O filtro de Bloom pode ser representado como um conjunto de $m$ bits. Quando o filtro está vazio, todos os bits estão com valor 0 (desligado). Para inserir um termo no filtro, é necessário passar o termo por uma quantidade $k$ de funções de *hashing*, em que o resultado de cada $k_i$ função indicará a posição de um *bit* que deve ter seu valor alterado para 1 (ligado). Por se tratar de uma estrutura de dados probabilística, o filtro tem uma taxa de falso-positivo. É possível através das Equações 01 e 02 definir valores $m$ e $k$ ótimos para que um filtro que acomode $n$ termos apresente uma taxa de falso-positivo $p$ aceitável (MITZENMACHER; UPFAL, 2005).

$$m = -\left(\frac{n \times \ln(p)}{(\ln(2.0))^2}\right) \quad (01)$$

$$k = \ln(2.0) \times \frac{m}{n} \quad (02)$$

Para cada arquivo indexado, um índice (o filtro de Bloom) é criado a partir do conjunto $H$ associado ao arquivo, em que a quantidade de itens no conjunto $H$ aponta um valor $n$ que possibilita calcular o tamanho $m$ que o filtro de Bloom deve ter. Para o modelo definido foi estabelecida uma taxa de falso-positivo de um em um milhão, com isso $p$ assume o valor de $10^{-6}$. Utilizando a Equação 02, pode-se definir um valor $k$ ótimo para o $p$ fixado. Então para um arquivo com uma lista de termos $H$ e $\forall h \in H$ são calculados $k$ *hashes* onde $k_i \neq k_j$ quando $i \neq j$, e para cada posição obtida como resultado de $k$ temos que o bit da posição correspondente é ligado no filtro de Bloom.

O resultado gerado pelo módulo de indexação é uma sequência de bits que representa o arquivo indexado, desta forma, é possível verificar se um termo provavelmente faz parte ou definitivamente não faz parte de um determinado arquivo, uma vez que o filtro de Bloom pode retornar falso-positivo, mas nunca falso-negativo (BLOOM, 1970).

Após esta etapa, o índice é gravado em arquivo e vai para a fila do módulo de sincronização, que posteriormente será enviado para o servidor remoto juntamente com o arquivo encriptado com a chave do usuário. Após isso, o arquivo estará disponível para consulta privada, pois o conteúdo do mesmo não estará visível para o servidor, e, além disso, para a realização das consulta, o servidor não tem visibilidade para o conteúdo de quais termos estão sendo buscados. Mais detalhes do processo de consulta são discutidos na subseção seguinte.

### 3.2. Consultas

Depois que um arquivo foi indexado e sincronizado com o servidor remoto, este passa a estar disponível para consultas através do Motor de Busca do servidor remoto. Com isto, um cliente pode fazer buscas sem a necessidade de sincronizar todo o conteúdo do servidor para o computador.

Para a realização das consultas, o usuário no

módulo cliente deve inserir os termos a serem pesquisados. O conteúdo textual da consulta passa pelo mesmo processo ilustrado na Figura 2: os termos da consulta são normalizados, reduzidos, filtrados por *stop words* e passam pelo procedimento de *hashing*. Como resultado, temos um conjunto *Q* que contém o *hash* de todos os termos consultados, este conjunto é encaminhado para o servidor que é o local em que a consulta é processada.

O servidor ao receber o conjunto de termos de consulta *Q* realizará a busca em *F*, que é o conjunto de todos os filtros de Bloom armazenados para o usuário. Para cada $Q_i$ é testado se provavelmente este está em algum $F_i$. Para cada correspondência encontrada, uma referência para o arquivo relacionado é inserida em um conjunto de resultados *R*. Ao final, o servidor retorna para o cliente o conjunto *R* em que cada $R_i$ provavelmente tem um ou mais dos termos pesquisados pelo usuário.

Para este modelo, a privacidade do usuário é mantida mesmo que os arquivos sejam armazenados remotamente em um servidor não confiável, pois o servidor não tem acesso ao conteúdo original das consultas, índices ou arquivos. Sendo assim, a única forma de obter acesso aos termos do índice seria através do método de força bruta, que é inviável em função do tempo computacional requerido para fazê-lo. O modelo faz uso de funções de *hashing*, criptografia dos arquivos, e pela característica probabilística do filtro de Bloom seria necessário muito tempo de computação para tentar minerar o conteúdo original dos índices, arquivos e consultas.

## 4. IMPLEMENTAÇÃO

Na implementação do modelo definido neste trabalho (ilustrado na Figura 1), tem-se que no Sistema de Arquivos uma rotina de encriptação de blocos é executada de maneira transparente ao usuário. Isto é feito através do *EncFS* (ENCFS, 2017), o qual é um sistema de arquivos encriptado e transparente para o usuário baseado no *Fuse* (SZEREDI. 2005). O *Fuse* é um módulo presente no *Kernel Linux*, com o qual é possível a criação e montagem de sistemas de arquivos sem a necessidade de manipular o *Kernel* para fazer o gerenciamento dos mesmos. Assim, toda vez que um arquivo é criado ou alterado, uma versão criptografada deste arquivo é automaticamente gerada por meio do *EncFS*.

Além disso, se o arquivo for indexável, isto é, se for um documento com formato conhecido tal como *pdf*, *doc*/ *docx*, *txt*, *rtf*, *md*, *cvs* ou *odt*, é utilizada a biblioteca *textract* (MALMGREN, 2014), para a extração dos termos textuais do documento para posterior processo de indexação.

No processo de geração do conjunto *H* de termos para indexação (processo detalhado na Seção 3.1), tem-se que os termos passam por um processo de *hashing* utilizando uma função de *hash* criptográfica denominada *MD5* (RIVEST, 1992). A quantidade distinta de termos usualmente presente em documentos e a não necessidade da resistência de colisões torna o uso da função de *hashing MD5* adequada para a implementação.

Já para a criação do filtro de Bloom no processo de indexação descrito na Seção 3.1, são necessárias *k* diferentes funções hash. Sendo assim, na implementação foram adotadas duas funções de *hashing* distintas: a *FNV* e a *Murmurhash3*. A primeira função utilizada é a *FNV*, já todas as outras *k-1* funções são a *murmurhash3* com diferentes parâmetros de *seed.* Cada valor de *seed* permite gerar resultados de hashing diferentes, pois para cada $i,j \leq k-1$ tal que $i \neq j$ tem-se que $murmurash3(termo,i) \neq murmurash3(termo,j)$. Desta forma é possível obter *k-1* resultados diferentes sem a necessidade de *k-1* implementações de funções *hash* distintas.

A implementação do modelo definido e discutido neste trabalho foi incluído em uma aplicação denominada *SSRS* (*Searchable Secure Remote Storage*). Nesta aplicação, o usuário pode montar um diretório remoto no computador, após isto, todos os arquivos que forem adicionados neste ponto de montagem serão encriptados e sincronizados com o servidor remoto. Se o arquivo for indexável será indexado e posteriormente consultas para o conteúdo do mesmo podem ser realizadas. Um exemplo de uso da aplicação *SSRS* é ilustrado na próxima seção deste trabalho.

## 5. EXEMPLO DE USO

Para utilizar a aplicação *SSRS* (Searchable Secure Remote Storage) o usuário deve possuir o endereço remoto do servidor e uma pasta local vazia, nesta pasta será montado o sistema de arquivos remoto. Após abrir a aplicação SSRS, o usuário clica no botão *"Add"* para adicionar um novo ponto de montagem (conforme ilustrado na Figura 3). Após isto, o usuário insere o endereço remoto do servidor e o caminho local para o ponto de montagem, este procedimento necessita ser feito uma única vez para cada ponto de montagem.

Após criar o ponto de montagem, o usuário seleciona o ponto de montagem e clica no botão "*Mount"* (Figura 3). A aplicação irá solicitar a senha do ponto de montagem e iniciará a rotina de montagem. Após isso, os dados pertencentes ao ponto de montagem estarão disponíveis no diretório especificado como *"Local Path"*, e também se torna possível a sincronização de novos arquivos inseridos ou modificados no ponto de montagem através do botão *"SYNC"* (Figura 3). Por fim, é possível desmontar um ponto de montagem através do botão "*Unmount*" (Figura 3), bem como é possível remover um ponto de montagem através do botão "*Remove"* (Figura 3).

Em relação à funcionalidade de busca por documentos, esta pode ser iniciada através do botão "*Find*" (Figura 3). Após clicar neste botão surge uma nova tela de busca com uma caixa de texto para inserção dos termos a serem pesquisados (esta tela está ilustrada na Figura 4). Na tela de busca (Figura 4) é possível realizar a consulta textual por documentos que estejam em um ponto de montagem previamente montado no dispositivo atual. Após realizar a consulta, os resultados são dispostos em uma lista de resultados e podem ser carregados diretamente pela aplicação.

**Figura 3. Tela para montagem de armazenamento remoto na aplicação SSRS.**

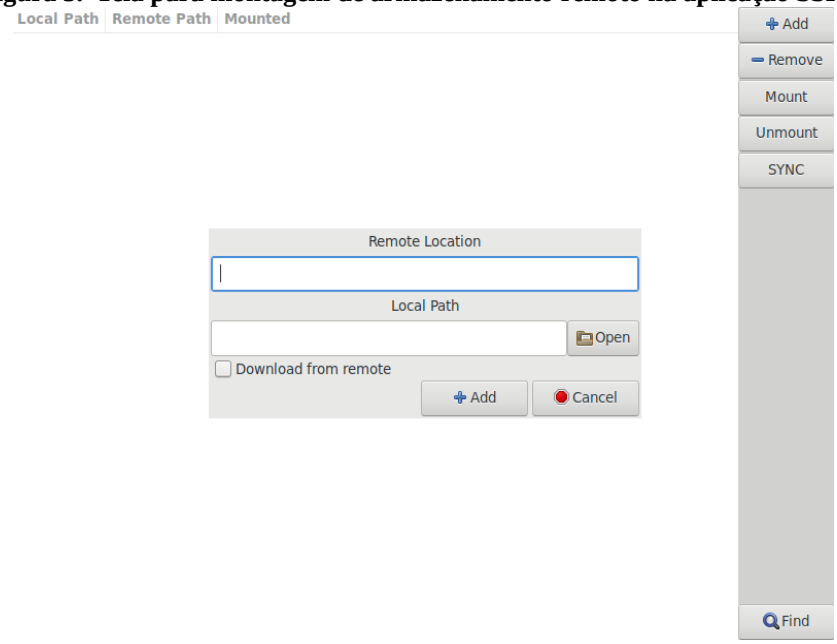
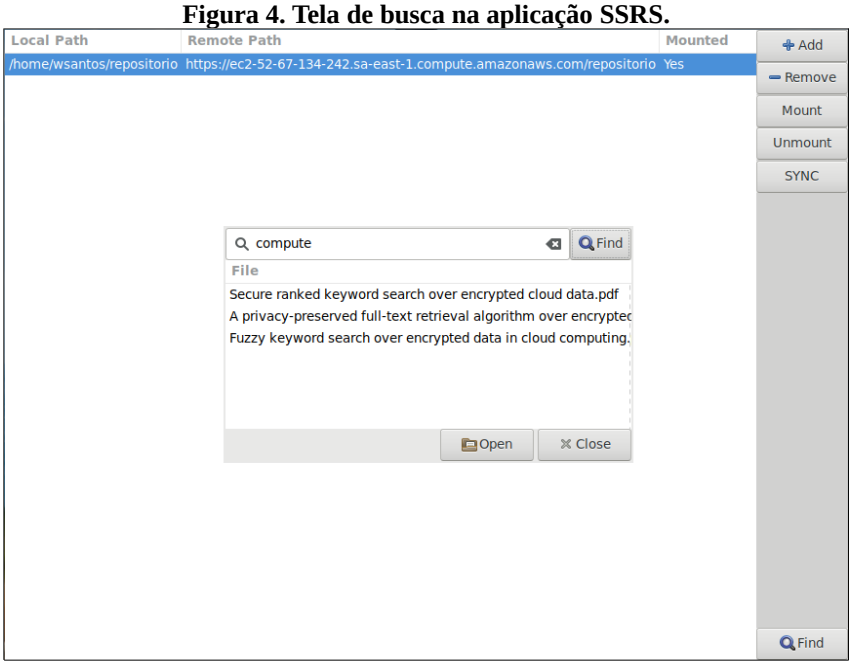



**Figura 4. Tela de busca na aplicação SSRS.**



## 6. AVALIAÇÃO EXPERIMENTAL

A implementação do modelo de indexação e recuperação privada de documentos definido neste trabalho apresentou-se funcional em todos os testes conduzidos tanto no cliente quanto no servidor, desta forma trazendo respostas corretas na indexação e consulta de arquivos. Além disso, um experimento foi conduzido com o objetivo de verificar e compreender o desempenho das estruturas de dados e algoritmos de indexação e recuperação no modelo definido neste trabalho. O experimento foi executado em um computador *Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz* com *8GB* de memória *RAM*.

Primeiramente um conjunto de documentos foi aleatoriamente gerado. Em cada documento um conjunto de termos unívocos foi aleatoriamente gerado, sendo que nenhum documento tem termos repetidos. Foram gerados cinco documentos, em que cada um tem 500, 1000, 2000, 4000 e 8000 termos unívocos respectivamente.

A primeira questão avaliada é a variação do tamanho do filtro de Bloom gerado no processo de indexação bem como o tempo de indexação em relação ao tamanho do documento, isto é, em relação à quantidade de termos que cada documento contém. Na Tabela 1 é ilustrada a relação entre a quantidade de termos indexados de cada documento e: o tamanho *m* do filtro de Bloom gerado; a quantidade *k* de funções de *hash* aplicadas na indexação de cada termo.

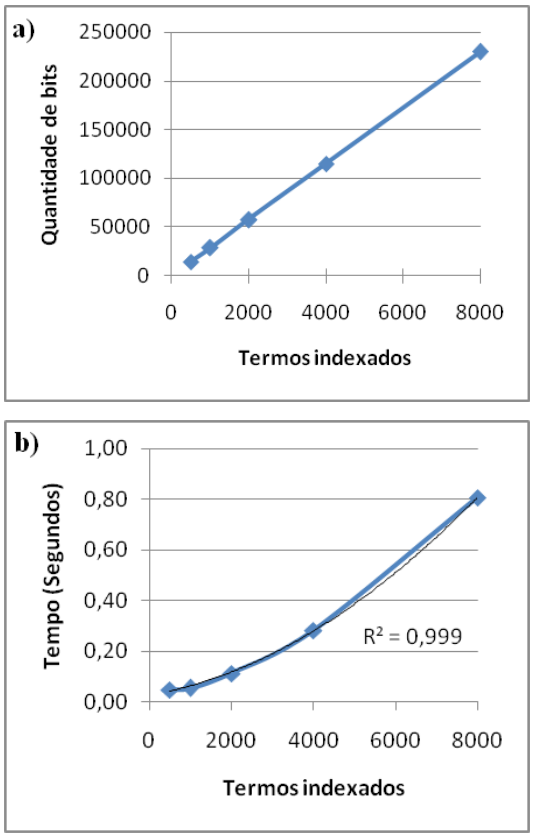
Vale ressaltar que, como em cada documento todos os termos são unívocos (nenhum termo se repete), após o documento passar pelo processo de Redução (Figura 2) para remover termos duplicados, tem-se que a quantidade de termos indexados é igual à quantidade original de termos na entrada.

**Tabela 1. Tamanho do filtro de Bloom e quantidade de funções de hash *vs* termos indexados.**

| Termos na entrada | Termos indexados (*n*) | Tamanho (bits) do filtro de Bloom (*m*) | Quantidade de funções de hash (*k*) |
|---|---|---|---|
| 500 | 500 | 14378 | 20 |
| 1000 | 1000 | 28756 | 20 |
| 2000 | 2000 | 57511 | 20 |
| 4000 | 4000 | 115021 | 20 |
| 8000 | 8000 | 230042 | 20 |

O gráfico da Figura 5 (a) mostra o crescimento do tamanho *m* do filtro de Bloom em *bits* em relação à quantidade de termos indexados. Quando a quantidade de termos indexados dobra, o tamanho do filtro aproximadamente dobra. Logo, quanto maior a quantidade de termos indexados do arquivo maior é o índice (o filtro de Bloom) gerado para o mesmo. Em contrapartida, a quantidade de funções de hash *k* varia de forma extremamente lenta, e no caso do experimento é constante para todas as entradas avaliadas (conforme indicado na Tabela 1).

**Figura 5. a) Tamanho do filtro de Bloom e b) Tempo de indexação.**




Na Tabela 2 são listados os tempos de indexação (em segundos) para cada um dos documentos indexados. Para cada documento, o processo de indexação foi realizado cinco (5) vezes. Sendo assim, o tempo de indexação indicado na Tabela 2 é a média destas cinco execuções. Como pode ser verificada, para todos os testes executados para cada documento, a variação de tempo (dispersão) entre as execuções foi baixa dado que o desvio padrão bem como o coeficiente de variação tem valor extremamente baixo. Sendo assim, consideram-se as execuções confiáveis para obtenção da média.

**Tabela 2. Desempenho do módulo de Indexação: número de termos indexados *vs* tempo em segundos.**

| Termos indexados (*n*) | Média do tempo de indexação (segundos) | Desvio padrão (segundos) | Coeficiente de variação (%) |
|---|---|---|---|
| 500 | 0,04779 | 0,00053 | 1,11% |
| 1000 | 0,05407 | 0,00047 | 0,88% |
| 2000 | 0,11260 | 0,00033 | 0,29% |
| 4000 | 0,28203 | 0,00109 | 0,39% |
| 8000 | 0,80714 | 0,00570 | 0,71% |

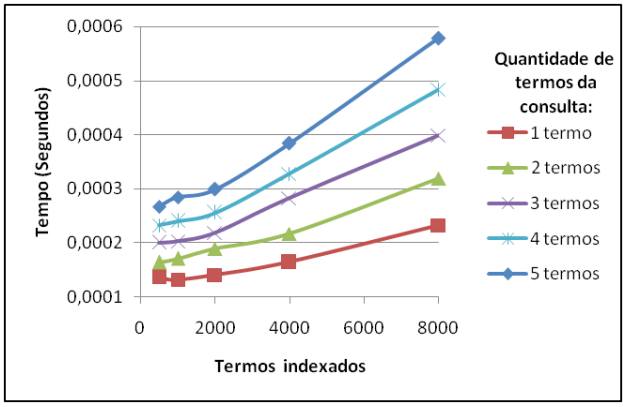
A variação do tempo de indexação em relação à quantidade de termos indexados também é mostrada no gráfico da Figura 5 (b). Conforme ilustrado no gráfico da Figura 5 (b), por regressão é possível determinar um comportamento de crescimento (indicado pela linha de tendência no gráfico) polinomial de ordem 2 (quadrático) do tempo de indexação em função da quantidade de termos indexados. A regressão tem coeficiente de determinação ($R^2$) dado por 99,97%, indicando um bom ajuste.

É importante ressaltar que no processo de indexação (ilustrado na Figura 2), os subprocessos de Separação, Normalização, Filtragem e Hash são executados linearmente sob cada termo do documento indexado, isto é, para cada processo é necessário passar uma vez em cada termo. Apesar disso, no caso do processo de Redução é necessário manter uma lista dos termos já encontrados para verificar se um termo é ou não repetido, este procedimento em pior caso não pode ser feito de maneira linear, o que contribui para o resultado do tempo de indexação apontado pelo gráfico da Figura 5 (b).

Já em relação ao tempo de consulta, o gráfico da Figura 6 mostra a média dos tempos de execução de consultas (cada consulta foi executada três vezes) em função da quantidade de termos (1, 2, 3, 4 e 5) da consulta e do tamanho do filtro de Bloom consultado.

Como pode ser observado no gráfico da Figura 6, o tempo de execução de uma consulta depende da quantidade termos da consulta bem como do tamanho do filtro de Bloom associado ao arquivo consultado. O tempo de consulta aumenta quando a quantidade de termos da consulta aumenta pois para cada termo é necessário executar *k* funções de *hash* e *k* verificações no filtro de Bloom do arquivo. Além disso, o tempo de consulta aumenta quando o tamanho do filtro de Bloom do arquivo aumenta, pois o tempo para carregamento do filtro de Bloom em memória principal torna-se maior.

**Figura 6. Tempo de consulta *vs* quantidade de termos da consulta e termos indexados**




## 7. CONSIDERAÇÕES FINAIS

Como discutido neste trabalho, o armazenamento seguro de documentos e a recuperação privada de informações se fazem cada vez mais necessários visando tanto a demanda pessoal quanto corporativa. Neste trabalho foi apresentado um modelo de indexação e recuperação de informação, o qual sob uma base de arquivos com conteúdo textual permite a busca de documentos que contenham termos pesquisados que são privados ao usuário bem como o conteúdo dos documentos não pode ser lido por serviços de armazenamento de terceiros.

Para alcançar isto, faz-se uso de métodos de criptografia do conteúdo dos documentos bem como a utilização de funções de *hashing* para criptografar o conteúdo das consultas. Os índices são criados através de filtros de Bloom, os quais são uma estrutura de dados probabilística para recuperação de informação. Apesar dos benefícios do ponto de vista de segurança, a utilização de filtros de Bloom faz com que o tempo de consulta seja linearmente dependente da quantidade de arquivos indexados, pois, quanto maior a quantidade de arquivos indexados (bem como a quantidade de termos em cada arquivo), maior é a quantidade de filtros de Bloom a serem testados no processo de consulta.

Na *World Wide Web*, temos que sistemas de busca em geral utilizam-se de índices invertidos para identificar quais documentos possuem termos informados através de consultas (HAWKING, 2006). Para cada um dos termos de um restrito vocabulário, um *índice invertido* armazena uma listagem de documentos tal que o termo ocorre. Isto permite aprimorar o tempo de execução de consultas, pois para um dado termo, todas as ocorrências do termo já foram pré-computadas e armazenadas no índice invertido.

Sendo assim, como trabalhos futuros almeja-se investigar a viabilidade de criação de índices invertidos privados de forma a aperfeiçoar os algoritmos de processamento de consultas para complexidade de tempo de execução sublinear em relação ao tamanho da fonte de dados indexada.

## REFERÊNCIAS BIBLIOGRÁFICAS

BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. **Commun. ACM,** ACM, New York, NY, USA, v. 13, n. 7, p. 422–426, jul. 1970. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/362686.362692>.

BONOMI, F. et al. An improved construction for counting bloom filters. **In: Proceedings of the 14th Conference on Annual European Symposium - Volume 14**. London, UK, UK: Springer-Verlag, 2006. (ESA'06), p. 684–695. ISBN 3-540-38875-3. Disponível em: <http://dx.doi.org/10.1007/11841036_61>.

ENCFS. **EncFS**. Web site. Disponível em: <https://vgough.github.io/encfs/>. Acessado em 10/11/2017.

HAWKING, D. **Web Search Engines: Part 2**, *Computer*, vol. 39, no. 8, pp. 88-90, Aug., 2006.

MALMGREN D. **textract**. 2014. Online. Disponível em: <https://github.com/deanmalmgren/textract>. Acessado em 03/06/2017.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. **Introduction to Information Retrieval.** New York, NY, USA: Cambridge University Press, 2008. ISBN 0521865719, 9780521865715.

MESSMER, S. et al. A novel cryptographic framework for cloud file systems and cryfs, a provably-secure construction. In: LIVRAGA, G.; ZHU, S. (Ed.). **Data and Applications Security and Privacy XXXI**. Cham: Springer International Publishing, 2017. p. 409–429. ISBN 978-3-319-61176-1.

RIVEST, R. The MD5 Message-Digest Algorithm. [S.l.], 1992. DOI: 10.17487/RFC1321.

SZEREDI, M. **FUSE**. 2005. Online. <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/plain/Documentation/filesystems/fuse.txt>.

VIEIRA, Claudia Simone. Computação em nuvem: fatores que influenciam a adoção pelas empresas no Brasil. **Tese (Doutorado em Administração de Empresas) - FGV** - Fundação Getúlio Vargas, São Paulo, 2017. Disponível em: <http://hdl.handle.net/10438/18024>.