
UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

BRUNA JULLY NEVES NUNES

DIEGO RODRIGUES DE SOUZA

# *OWNCRYPT*: UM APLICATIVO *WEB* DE CRIPTOGRAFIA DE DADOS NO CLIENTE PARA O *OWNCLOUD*

Belém
2017

BRUNA JULLY NEVES NUNES
DIEGO RODRIGUES DE SOUZA

# *OwnCrypt*: Um aplicativo *web* de criptografia de dados no cliente para o *OwnCloud*

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação, pela Universidade Federal do Pará.

Orientador: Prof. Dr. Roberto Samarone dos Santos Araújo.

Belém
2017

BRUNA JULLY NEVES NUNES
DIEGO RODRIGUES DE SOUZA

# *OWNCRYPT*: UM APLICATIVO *WEB* DE CRIPTOGRAFIA DE DADOS NO CLIENTE PARA O *OWNCLOUD*

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação, pela Universidade Federal do Pará.

Orientador: Prof. Dr. Roberto Samarone dos Santos Araújo.

Aprovado em: ___/___/2017

Conceito: _____

Banca Examinadora

_____
Prof. Dr. Roberto Samarone dos Santos Araújo
Orientador-UFPA

_____
Profª. Dra. Regiane Silva Kawasaki Francês
Professora- UFPA

_____
Profª. Dra. Marcelle Pereira Mota
Professora- UFPA

*Dedicamos este trabalho à memória daqueles amigos que não tiveram a chance de chegar até aqui como nós.*

**Agradecimentos**

Agradeço imensamente ao meu pai Luca, que além de ser meu maior incentivador, também é meu grande amigo, aquele que vibra com cada conquista, sem ele eu não teria conseguindo chegar até o fim. À minha mãe Clau (*in memorian)* onde ela estiver, sei que está comemorando esta conquista junto comigo.

Aos meus familiares (tios, primos, avós) pelas palavras de apoio e incentivo e em especial ao meu avô Aluísio (*in memorian),* que mesmo após sua perda recente e dolorosa, me fez ter força para continuar, pois sei o quão orgulhoso ele iria estar.

Ao meu amigo-irmão-parceiro Diego Rodrigues, por todos os anos de amizade, de parceria, por ter aguentado os momentos de angústia, ansiedade e estresse e em meio a tantas dificuldades por nunca ter desistido da caminhada na construção deste trabalho. Obrigada por tudo! E não menos importante, Karol Salgado, por ter tido paciência e ter aturado nossos estresses e frustrações, por ter se doado por nós e sempre ter feito o possível para que conseguíssemos concluir este trabalho.

Aos gordos (Hans e Wesley), se não fosse o nosso "quinteto" não teríamos conseguido dar conta de tudo o que tinha que ser feito para passar nos semestres. Vicente pelos momentos de risada e toda a parceria e cumplicidade que o Clube tem.

Aos amigos de infância: Aline Lobo por todas as conversas, as brincadeiras, os conselhos e por mesmo que agora esteja distante continua se fazendo presente (*You know in the end, I will always be there!*); Bel Lobato pelo companheirismo de sempre, seja nos momentos felizes, nos tristes, seja para compartilhar problemas ou apenas para perguntar se está tudo bem. Rodrigo Guimarães pelo compartilhamento de experiências sobre a área e por tudo o que nossa amizade representa.

À Karlena, por mesmo de longe, sempre me incentivar e por não me deixar desistir! Obrigada por *"Never give up on me!"*

Aos amigos que a UFPa me deu: Tathy e Laíza, por todos os auxílios nos momentos de angústia, por celebrar junto os momentos de felicidade e pela presença constante nos momentos de dificuldade.

E ao orientador professor Dr. Roberto Samarone, pela orientação, pela paciência e por todo o conhecimento compartilhado.

Bruna Nunes.

**Agradecimentos**

Agradeço aos meus pais, Dona Eliana e Seu Joaci, que sempre confiaram em mim e me incentivaram a continuar sempre, vocês são a base do meu caráter.

Serei eternamente grato à minha amada Ana Karolina, cujo companheirismo e amor me manteve de pé durante os períodos mais complicados dessa jornada e não me deixou desistir.

E meu muito obrigado aos amigos que conheci em todos esses longos anos da graduação que me ensinaram que juntos conseguimos mudar as coisas para melhor e que uma boa risada ajuda a aliviar o peso da vida. Assim como aos parceiros do "Clube" que dividem o peso da graduação e da vida fora da universidade dia após dia.

Um agradecimento especial aos amigos André Neto e Iuri Raiol, que passaram horas a fio arrumando código comigo. Valeu pessoal, isso nunca será esquecido.

Por fim, valeu maninha Jully, por sempre permanecer do meu lado. Amiga, parceira, irmã. E ao nosso orientador Prof. Dr. Roberto Samarone, por ter sido mais que um orientador para nós, sempre demonstrando preocupação para nós fazermos o melhor possível.

Diego Rodrigues de Souza

*Eu vim de longe para encontrar o meu caminho*
*tinha um sorriso e o sorriso ainda valia*
*achei difícil a viagem até aqui*
*mas eu cheguei, mas eu cheguei*

*Eu vim por causa daquilo que não se vê*
*vim nu, descalço, sem dinheiro e o pior*
*achei difícil a viagem até aqui*
*mas eu cheguei, mas eu cheguei*

*Eu tive ajuda de quem você não acredita*
*tive a esperança de chegar até aqui*
*vim caminhando, aqui estou, me decidi*
*eu vou ficar, eu vou ficar*

*Antiga canção escoteira*
*(Autor desconhecido)*

# Sumário

## Lista de Figuras

## Lista de Quadros

## Lista de Siglas

TI – Tecnologia da Informação

NIST – *National Institute of Standards and Technology*

SaaS – *Software as a Service*

PaaS – *Platform as a Service*

IaaS – *Infrastructure as a Service*

DES- *Data Encryption Standard*

AES – *Advanced Encryption Standard*

PGP – *Pretty Good Privacy*

RNP – Rede Nacional de Ensino e Pesquisa

IDEA – *Internacional Data Encryption Algorithm*

FTP – *File Transfer Protocol*

SSL – *Secure Socket Layer*

**Resumo**

Atualmente a maioria das nuvens de armazenamento criptografam os dados antes de armazena-los. No entanto, como essas nuvens também mantém as chaves criptográficas utilizadas no processo de criptografia, elas podem facilmente ter acesso ao conteúdo armazenado. Visando mitigar esse problema, este trabalho apresenta o *OwnCrypt*. Um aplicativo para o *frontend OwnCloud* que permite a criptografia de dados no computador do usuário, antes do envio para a nuvem.

**Palavras-chave**: Computação em nuvem, nuvem, segurança, armazenamento de dados, criptografia.

***Abstract***

*Currently most storage clouds encrypt the data before it is stored. However, because these clouds also keep the cryptographic keys used in the encryption process, they can easily access the stored content. In order to mitigate this problem, this paper presents OwnCrypt. An application for the OwnCloud frontend that allows the encryption of data on the user's computer before sending to the cloud.*

***Keywords****: Cloud computing, cloud, security, data storage, encryption*

# 1 INTRODUÇÃO

A computação em nuvem surgiu com a proposta de redução dos custos de uma infraestrutura computacional; seja custo na aquisição de equipamentos ou até mesmo custos com a manutenção. Ela possibilitou a criação de uma infraestrutura de TI (Tecnologia da Informação) onde os usuários não precisam instalar, configurar ou atualizar os sistemas, deixando estas responsabilidades nas mãos de terceiros. Sendo assim, a computação em nuvem é um novo modelo de computação que permite ao usuário final acessar uma grande quantidade de aplicações e serviços em qualquer lugar (PEDROSA; NOGUEIRA, 2011).

O modelo de computação em nuvem foi desenvolvido com o objetivo de fornecer serviços de fácil acesso, baixo custo e com garantias de disponibilidade e escalabilidade (MOREIRA; SOUZA, 2010). Através de tal facilidade de acesso é possível haver uma redução na aquisição da infraestrutura que será utilizada, já que seus serviços são fornecidos sob demanda.

De acordo com um estudo realizado por uma revista de segurança, a KPMG (2014) Internacional, entre os anos 2012 e 2014, foi notado que as vantagens de adotar a computação em nuvem vão além dos baixos custos, incluem também flexibilidade, inovação e melhor controle operacional. (KPMG INFO, 2014)

Ainda segundo esta pesquisa, outro dado interessante aborda a visão da nuvem como um facilitador das operações internas. No ano de 2012, apenas 15% dos executivos possuíam essa perspectiva, hoje esse número abrange cerca de 42%. Além disso, outra vantagem citada pelos entrevistados foi o aumento da produtividade em 54% dos seus colaboradores (SOCBLOG, 2015).

Em decorrência das várias vantagens que a computação em nuvem pode fornecer, sua utilização vem sofrendo um aumento expressivo e as perspectivas para o futuro são bem favoráveis. De acordo com a *Forrester Research*, o mercado de nuvem é estimado para chegar a $191 bilhões para 2020, uma subida significativa do total de $58 bilhões em 2013. (KPMG INFO, 2014).

Uma das formas de computação em nuvem é o armazenamento de dados. Nuvens de armazenamento de dados são uma forma simples e prática de guardar arquivos sem a necessidade de investir em equipamentos. Além disso, ela possibilita o acesso aos dados de qualquer lugar necessitando apenas dispor de conexão com a *internet*.

Atualmente existe no mercado diversas tecnologias de armazenamento de dados em nuvem, como o *Google Drive*, *Amazon Web Service*, *DropBox*, por exemplo, onde os dados são salvos em um ou mais servidores cuja localização física exata é desconhecida.

Apesar das vantagens, facilidades e economia que o armazenamento de dados em nuvem pode proporcionar, ainda há questões relacionadas à segurança que preocupam os usuários do serviço como, por exemplo, o acesso indevido aos dados por terceiros. Mecanismos criptográficos, no entanto, podem ajudar na proteção evitando assim que alterações possam ser realizadas nos dados armazenados na nuvem.

Observa-se a necessidade do usuário, quando da utilização da tecnologia mencionada, de obter garantia que o seu arquivo está armazenado de forma segura, ou seja, íntegro e não acessível a terceiros sem sua prévia autorização. Nesse sentido, o trabalho em questão visa primeiramente realizar um estudo sobre as ferramentas de código aberto mais relevantes para o armazenamento de dados em nuvem e comparar tais ferramentas entre si. Em particular, os *frontends* de armazenamento de dados. Além disso, também apresenta um aplicativo para uma dessas ferramentas que possibilita criptografia de dados.

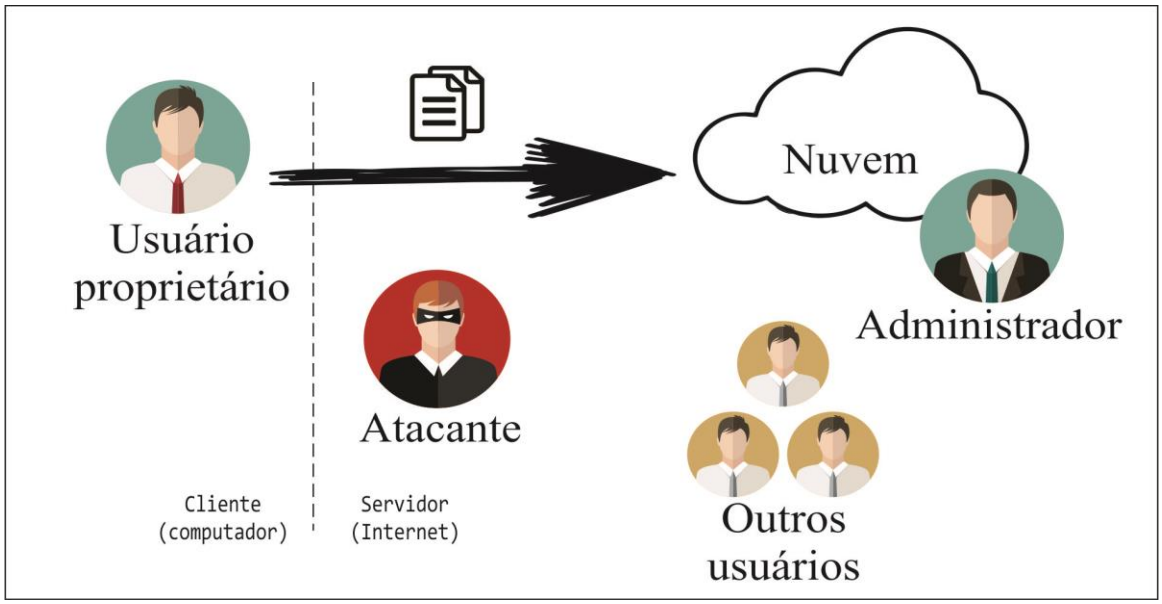
## 1.1 Motivação

Diante dos dados apresentados que comprovam o aumento significativo do uso das nuvens de armazenamento de dados e a tendência deste aumento ser continuo nos próximos anos, faz-se necessário um estudo sobre as alternativas de tecnologias de armazenamento.

Juntamente com o aumento do uso desta tecnologia, cresce também a preocupação dos usuários com a segurança deste serviço, principalmente quando se trata de arquivos pessoais, pois este envio é feito para servidores de terceiros em algum lugar na *internet*, os quais se responsabilizam por sua guarda. Considerando, aspectos de segurança e privacidade, levanta-se o seguinte questionamento: quem, efetivamente, possui acesso aos dados armazenados na nuvem?

Para responder tal questionamento, é necessário primeiramente identificar os possíveis agentes no processo de envio de um arquivo para a nuvem de armazenamento, são eles: o usuário proprietário do arquivo, outros usuários da nuvem que podem vir a ter acesso ao arquivo, o administrador da nuvem e possíveis atacantes.

*Figura 1: Fluxo de envio de arquivo para nuvem e seus possíveis agentes.*




Fonte: Elaborada pelos autores

Portanto, em resposta à pergunta anterior, durante este fluxo espera-se que apenas o usuário proprietário possua acesso ao arquivo. A maioria dos provedores de nuvem como a *Amazon e o Google* possuem um sistema de criptografia no servidor que visa impedir que os usuários da nuvem acessem os arquivos uns dos outros, no entanto, não existem garantias de que os administradores desses serviços não acessem os dados que estão armazenados sob sua responsabilidade já que as chaves utilizadas para criptografar os dados no servidor estão armazenadas sob poder do próprio administrador.

O acesso indevido pode ser realizado também por atacantes externos, *hackers*, por exemplo, através da exploração de falhas na segurança e até mesmo violação de senha de usuário.

Dessa forma, este trabalho trará benefício para a comunidade de usuários que utilizam nuvens de armazenamento de dados, através da criação de um *app* que permita que somente o proprietário tenha acesso ao arquivo que deseja salvar na nuvem. Vale ressaltar que esta solução é especificamente para a plataforma *OwnCloud*, uma plataforma *open-source* que será apresentada nos capítulos seguintes.

## 1.2 Objetivos

O objetivo geral deste trabalho é o desenvolvimento de um aplicativo de criptografia de dados no cliente para a nuvem de armazenamento de dados *OwnCloud*.

Os objetivos específicos deste trabalho são:

- Estudar as principais tecnologias de *frontend* de código aberto de armazenamento de dados em nuvem;
- Comparar as ferramentas de *frontend* estudadas;
- Estudar mecanismos criptográficos que possibilitem a introdução de criptografia de dados em um dos *frontend* estudados;

## 1.3 Trabalhos Relacionados

Na literatura podem ser encontrados vários trabalhos que demonstram a vantagem da utilização de computação em nuvem e sobre a utilização da criptografia para proteção dos dados.

Um exemplo destes trabalhos é o de Pedrosa e Nogueira (2011) onde são apresentadas as vantagens que um modelo de serviços de nuvem pode trazer, tal como o modelo de pagamento pelo uso que evita desperdício de recursos computacionais e, graças à sua escalabilidade é possível disponibilizar os recursos de acordo com a necessidade do usuário.

Outro trabalho nesse sentido é a Cartilha de Segurança para *Internet* do cert.br que relaciona algumas vantagens trazidas pelo uso da criptografia. Utilizando a criptografia o usuário pode:

- Proteger os dados sigilosos armazenados em seu computador;
- Proteger os *backups* contra acessos indevidos;
- Proteger as comunicações e transações realizadas pela *Internet.*

Em Salles et al. (2013) o Projeto SEGUREGENMOD desenvolveu um aplicativo que deixa o usuário mais seguro e menos vulnerável a monitoramento e roubo de dados efetuados tanto por terceiros quanto pelas prestadoras de serviços dos aplicativos. Verificou-se que a maioria das tecnologias de armazenamento de dados possui uma comunicação segura entre o cliente e o servidor, ou seja, realiza criptografia, porém essa chaves ficam sob posse do próprio servidor.

Duas tecnologias como o *SpiderOak* e o *Wuala* oferecem criptografia de dados no cliente, ou seja, as chaves criptográficas de ambos aplicativos não são enviadas para o servidor, e desta forma as informações enviadas não serão acessíveis pelos administradores da nuvem.

Outro trabalho relacionado com a importância da utilização da criptografia é o desenvolvido por Dias e Sakude (2008) no ITA (Instituto Tecnológico de Aeronáutica) onde o *software Itacripto* requer que o usuário utilize uma chave pública antes do arquivo ser enviado para o método de criptografia e que utilize sua chave privada para assinar o arquivo visando assim manter a autenticidade da informação.

No artigo *Desktop Client* do *StackSync* verifica-se a existência de um aplicativo que realiza a criptografia de dados no cliente onde a chave utilizada para a criptografia dos arquivos fica sob absoluta responsabilidade do usuário, característica essa que impede que as informações sejam roubadas ou acessadas por pessoas não-autorizadas. Tal método é utilizado como base para o desenvolvimento do aplicativo deste trabalho.

## 1.4 Estrutura

Este trabalho está estruturado da seguinte forma:

- Capítulo 2 - Fundamentação teórica. Este capítulo aborda os conceitos necessários para a realização deste trabalho, tais como, computação em nuvem, armazenamento de dados em nuvem; segurança da informação, criptografia e seus tipos.
- Capítulo 3 - Tecnologias de Armazenamento de Dados em Nuvem. Este capítulo descreve o cenário geral de uma nuvem de armazenamento de dados. Também são apresentadas as principais tecnologias de *frontend* utilizadas para acessar os dados armazenados na nuvem, tais como: *Pydio, StanckSync* e *OwnCloud*; tecnologias utilizadas para *backend* como: *OpenStack Swift, Amazon S3* e o FTP e por fim, realiza um comparativo entre os *frontends* apresentados abrangendo aspectos de segurança.
- Capítulo 4 – *OwnCrypt*. Este capítulo apresenta o desenvolvimento de um aplicativo *web* para criptografia de dados no cliente. Este aplicativo é desenvolvido exclusivamente para a plataforma do *OwnCloud*, e utiliza certificados digitais PGP para a criptografia dos dados. Também são apresentadas as tecnologias utilizadas para o desenvolvimento da aplicação, assim como as bibliotecas utilizadas.
- Capítulo 5- Conclusão e trabalhos futuros. Este capítulo condensa as considerações finais sobre o trabalho realizado, os resultados obtidos e limitações que este trabalho encontrou. Por fim, são sugeridos alguns possíveis trabalhos futuros baseando-se neste.

# 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos referentes à computação em nuvem. Ele também descreve o referencial teórico utilizado para o desenvolvimento deste trabalho, tais como, modelos de serviço da computação em nuvem, armazenamento de dados em nuvem, segurança da informação e criptografia de dados.

## 2.1 Computação em Nuvem

De acordo com o NIST no artigo *The NIST Definition of Cloud Computing* ( 2011), computação em nuvem é:

> um modelo que permite de forma conveniente, o acesso à rede sob demanda para um conjunto compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente provisionados e lançados com o mínimo de esforço de gestão ou a interação de um prestador de serviços.

Este tipo de computação pode ser considerado como uma possível solução para o grande crescimento de usuários que utilizam os serviços de TI. Os serviços passam a possuir um melhor desempenho e podem ser acessados por diferentes meios desde que os mesmos possuam conexão com a *internet.*

A computação em nuvem proporciona a flexibilização dos recursos computacionais facilitando assim a adição ou até mesmo modificação dos recursos, e também fornece escalabilidade e disponibilidade tanto em nível de *hardware* quanto de *software.* Outro objetivo deste modelo computacional é a redução no custo e na aquisição de equipamentos para a infraestrutura.

A computação em nuvem tem como proposta um modelo onde paga-se apenas por aquilo que for utilizado. Os computadores serão considerados como meios intermediários na conexão, pois os programas e as informações ficarão armazenados na nuvem e podem ser acessados a qualquer hora e em qualquer lugar.

A arquitetura da computação em nuvem pode ser dividida em: *hardware,* infraestrutura, plataforma e aplicação. A camada de *hardware* é responsável pelos recursos físicos: servidores e roteadores e é responsável também pela configuração dos *hardwares;* a camada de infraestrutura é a composição formada pelos discos de armazenamento, pelas máquinas virtuais e pelos recursos computacionais; a plataforma é responsável pela execução dos aplicativos dos usuários e a camada de aplicação é a que interage com o usuário, pois é ela quem fornece as aplicações da nuvem. [adaptado de (Vecchiola et al. 2009)]

De acordo com o NIST (2011), temos como características essenciais da computação em nuvem:

- Autoatendimento sob demanda: o usuário pode usufruir das funcionalidades computacionais sem a necessidade de interação humana com o provedor de serviço, ou seja, o provedor identifica as necessidades do usuário, podendo assim, reconfigurar automaticamente *hardwares* e *softwares,* modificações estas que devem ser apresentadas ao usuário de forma transparente. Esta característica está ligada à capacidade de provisionamento de recursos de forma automatizada.
- Acesso amplo à rede: os recursos computacionais são acessados através da *internet,* sem a necessidade de o usuário modificar o ambiente de trabalho do seu dispositivo. Esta característica está relacionada à capacidade de acesso, por diversos dispositivos, a recursos de rede computacional.
- Repositório de recursos: os recursos computacionais (físicos ou virtuais) do provedor são divididos em *pools* para que possam atender aos múltiplos usuários simultaneamente. Estes recursos são alocados e realocados dinamicamente de acordo com a demanda dos usuários, e eles não precisam saber a localização física dos recursos computacionais. Está relacionado à capacidade do provedor da nuvem de agrupar e mover recursos (físicos ou virtuais) para acomodar as necessidades de expansão e demanda do cliente.
- Elasticidade rápida: refere-se à capacidade de rápido provisionamento de recursos de acordo com a demanda, as funcionalidades em alguns casos podem ser liberadas automaticamente caso haja necessidade devido à demanda.
- Serviços mensuráveis: os serviços em nuvem controlam e monitoram automaticamente os recursos necessários para cada tipo de serviço, tais como, armazenamento, processamento e largura de banda. Refere-se à capacidade de medir a utilização de recursos de acordo com o serviço oferecido.

### 2.1.1 Modelos de Nuvens

As nuvens computacionais podem ser públicas, quando os seus recursos são disponibilizados ao público em geral; privadas, quando os recursos são geridos exclusivamente por uma organização; ou híbridas, quando apresentam características de mais de um modelo de nuvem ao mesmo tempo.

A nuvem pública pode pertencer à uma organização que será responsável por oferecer o serviço em nuvem deixando disponível para todas as pessoas, neste modelo não há restrição de acesso aos serviços, ou seja, pode ser acessada por qualquer usuário.

Uma nuvem privada é um tipo de nuvem desenvolvida para atender demandas específicas, geralmente utilizada por uma organização; pode estar hospedada localmente ou pode ser acessada remotamente. É o modelo de nuvem criado para uma empresa ou simplesmente para apenas um usuário, neste modelo, políticas de acesso são utilizadas.

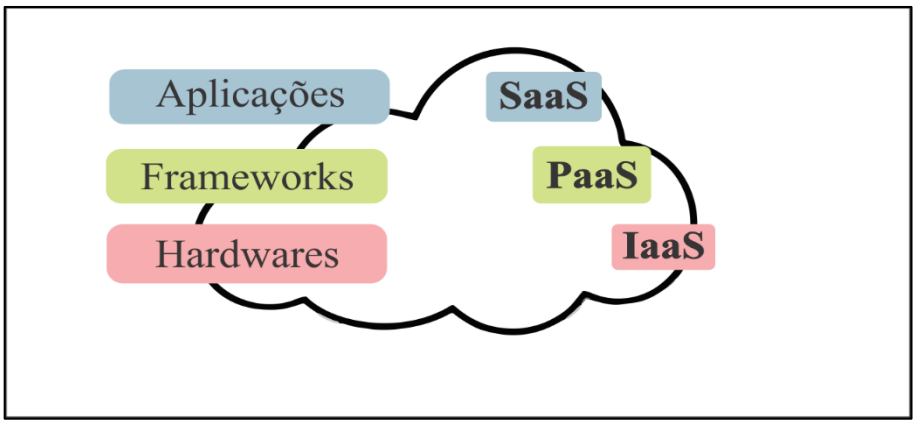
Segundo Jansen e Grance (2011):

> Uma nuvem pública é aquela em que a infraestrutura e recursos computacionais são disponibilizados ao público em geral através da *Internet*. Ela é de propriedade privada e é operada por um provedor de nuvem que realiza a entrega de serviços em nuvem para os consumidores e, por definição, é externo às organizações de consumidores. Na outra extremidade do espectro estão nuvens privadas. Uma nuvem privada é aquela em que o ambiente de computação é operado exclusivamente para uma única organização. Pode ser gerido pela organização ou por um terceiro, e pode ser hospedado dentro do centro de dados da organização ou fora dela. Uma nuvem privada tem o potencial de dar à organização um maior controle sobre a infraestrutura e aos recursos computacionais.

### 2.1.2 Modelos de Serviços em Nuvem

De acordo com Oliveira (2014), o padrão arquitetural da computação em nuvem é formado por três camadas: a camada de aplicação (SaaS), a camada de *frameworks* (PaaS) e a camada de *hardware* (IaaS) descritos a seguir:

*Figura 2: Modelos de serviços em nuvem*




Fonte: Elaborada pelos autores

- **SaaS - *Software* como Serviço**

Entre as principais características deste modelo estão a possibilidade de acesso dos dados via *web* e o gerenciamento dos mesmos de forma centralizada, onde os usuários só tem acesso às aplicações não possuindo acesso à infraestrutura.

Este modelo de computação em nuvem é considerado completo, pois todos os seus serviços são gerenciados pelo provedor e as aplicações presentes neste modelo são utilizadas pelo usuário final. O *Google Docs* é um exemplo de aplicação de um *software* como serviço.

O modelo de *software* como serviço baseia-se na ideia de fornecer ao consumidor um determinado serviço que está sendo operado e mantido na nuvem como execução no dispositivo do usuário (DIÓGENES, 2012).

De acordo com Borges et al. (2010) o SaaS possui as seguintes vantagens:

- Os aplicativos são disponíveis em qualquer dispositivo computacional, em qualquer lugar e a qualquer hora;
- Só há cobrança pelo que é utilizado não existindo taxas de licença;
- Atualizações ficam sob responsabilidade de quem provém o serviço, o usuário não necessita baixar ou instalar nada;
- A infraestrutura é escalável de acordo com a necessidade do usuário.

- **PaaS - Plataforma como Serviço**

Neste modelo as empresas que utilizam o serviço não precisam se preocupar com o gerenciamento da infraestrutura tanto em relação a *hardware* quanto em relação à atualização de sistemas operacionais, as empresas preocupam-se apenas com o gerenciamento das aplicações.

É uma plataforma criada para hospedar e gerenciar os *softwares*, sua utilização é indicada em serviços que necessitam utilização de banco de dados ou outros serviços de armazenamento. Atividades como desenvolver, compilar e testar passam a ser executadas diretamente na nuvem, um exemplo de utilização deste modelo é o *Google App Engine*.

Apesar de este modelo ser mais flexível, o nível de abstração da manutenção dos serviços também continua transparente. [...] O contratante do serviço também não tem acesso aos componentes da infraestrutura da rede (DIÓGENES, 2012).

Os recursos existentes na plataforma são acessados através de *frameworks*, devido ao fato do usuário não ter acesso à infraestrutura.

Abaixo estão listadas vantagens da utilização deste modelo:

- Menor investimento em relação à infraestrutura;

- Atualizações e novas funcionalidades são disponibilizadas de imediata;
- Suporte ágil à solução de problemas;
- A empresa contratante dos serviços não precisa se preocupar com a manutenção dos sistemas;
- Aumento da disponibilidade e segurança dos dados.

- **IaaS - Infraestrutura como Serviço**

Este modelo em nuvem fornece grande flexibilidade e fornece também controle no gerenciamento de recursos virtualizados e o valor cobrado é de acordo com os serviços e recursos contratados. A infraestrutura deste modelo é composta por plataformas de desenvolvimento, permite acesso aos recursos da rede e também fornece espaço para que os dados sejam armazenados. Um exemplo da utilização deste modelo é o *Amazon Elastic Cloud Computing* (EC-2).

De acordo com DIÓGENES (2012):

> A infraestrutura em questão inclui a disponibilização de rede, dispositivo de armazenamento e, diferentemente dos outros modelos, neste o contratante tem acesso ao sistema operacional, do ponto de vista de instalação, configuração e manutenção.

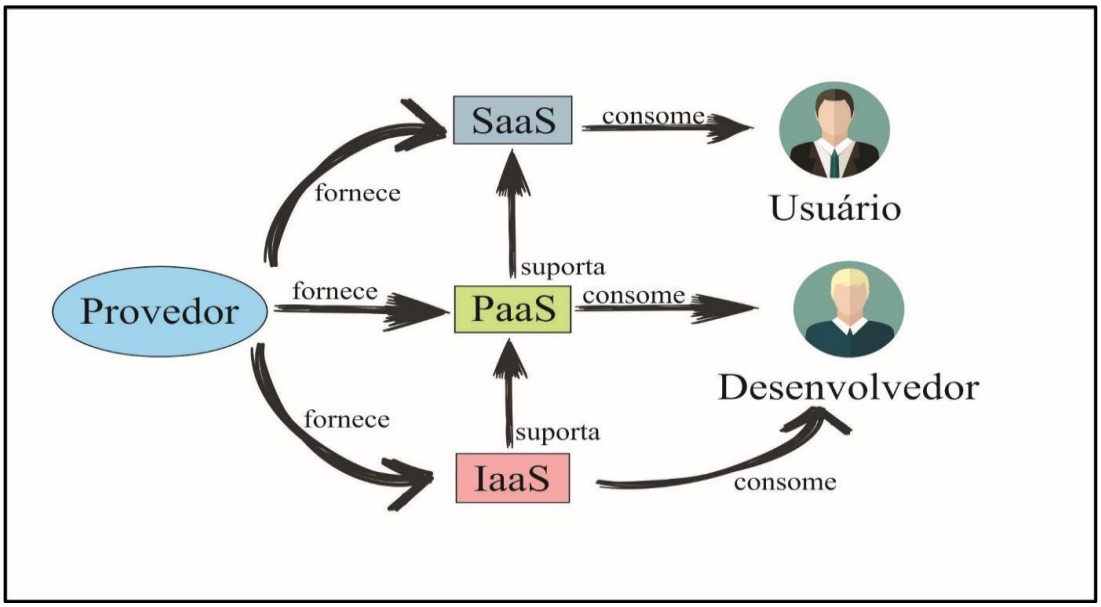
Em seus estudos, Borges et al. (2010) mostra como vantagens de trabalhar com IaaS:

- Redução de investimentos em *hardware*;
- Eliminação de custos com segurança e manutenção;
- Otimização do desempenho;
- Liberação do espaço físico;
- Flexibilidade para ampliar ou reduzir capacidade de processamento e/ou armazenamento.

- **Relação entre os Modelos de Serviços em Nuvem e seus Atores**

Na figura a seguir, verifica-se a relação entre os modelos de computação em nuvem onde há a existência do provedor, desenvolvedor e usuário. O provedor é responsável por disponibilizar, gerenciar e monitorar a estrutura da nuvem; o desenvolvedor é responsável por fornecer os serviços e recursos que serão utilizados pelo usuário.

*Figura 3: Relação entre os modelos de computação em nuvem*




Fonte: Elaborada pelos autores.

## 2.2 Segurança da Informação

De acordo com Castro (2011), “a segurança da informação nada mais é do que garantir a integridade e proteção das informações de uma organização. ”

A segurança da informação objetiva assegurar que os dados sejam confiáveis e para isso é necessário observar três aspectos considerados os pilares da segurança da informação: confidencialidade, integridade e disponibilidade.

Vale ressaltar que é prescindível a presença simultânea dos três aspectos para classificar uma informação como segura, já que o fator determinante é o tipo da informação tratada.

- **Confidencialidade**

O conceito de confidencialidade remete à ideia de que toda informação somente deve ser acessada pelo proprietário e pelos usuários autorizados pelo proprietário, de forma que o acesso indevido não retorne informação legível, seja durante a transmissão da informação ou durante o seu armazenamento.

A confidencialidade faz referência à prevenção da exposição da informação de forma não autorizada. Confidencialidade em sistemas de computação em nuvem, de acordo com (BENATALLAH, 2012) está relacionado com as áreas de direito de propriedade intelectual, canais secretos, análise de tráfico, criptografia e inferência.

Na computação em nuvem, a confidencialidade deve agir quando os arquivos estiverem sendo transferidos, pois a mesma deve ser feita de forma segura, por uma conexão criptografada por exemplo, impedindo assim seu acesso por usuários não autorizados.

- **Integridade**

O conceito de integridade passa a ideia de que os dados que são enviados serão recebidos pelo destinatário da mesma forma que quando saíram do remetente, isto é, sem modificações.

Sobre o fato da integridade ser um requisito para a existência de segurança, segundo (BENATALLAH, 2012) “este princípio especifica que os dados devem ser protegidos contra tais riscos: uma perda ou acesso não-autorizado, uso, modificação, destruição ou divulgação de dados. ”

Portanto, para que a integridade seja mantida espera-se que mesmo haja interceptação da informação, não seja possível alterar seu conteúdo, assim mudanças não deverão ser realizadas por pessoas que não estejam autorizadas.

- **Disponibilidade**

A disponibilidade garante que o serviço esteja funcionando corretamente sempre que for requisitado. O serviço deve estar disponível sob qualquer condição, até mesmo se houver ataques como a negação de serviço, por exemplo, que torna o sistema indisponível para fornecimento dos serviços. (BENATALLAH, 2012)

Problemas ocasionados pela quebra da disponibilidade:

- Sistemas ficam fora do ar;
- Ataques de negação de serviço (*denial of service);*
- Perda de documentos;
- Perda de acesso à informação.

### 2.2.1 Segurança em Computação em Nuvem

A computação em nuvem vem sendo considerada como a nova geração da Tecnologia da Informação, mas para a sua utilização é necessário que haja garantia de segurança nos serviços que são fornecidos.

Assim como qualquer modelo computacional, a computação em nuvem também enfrenta algumas dificuldades relacionadas às questões de segurança, que se não forem devidamente verificadas, poderão acarretar em sérios problemas tanto para as empresas contratantes dos serviços quanto para os usuários.

Práticas de segurança e privacidade implicam no monitoramento de ativos do sistema de informações da organização e na implementação de políticas, normas, procedimentos, controles e diretrizes que são usados para estabelecer e preservar a confidencialidade, integridade e disponibilidade dos recursos do sistema de informação (JANSEN; GRANCE, 2011).

## 2.3 Criptografia

Do grego: *kryptós,* escondido e *gráphien,* escrita, entende-se por criptografia o estudo de métodos e princípios que visam transformar uma informação legível em outra não legível, a qual não é facilmente lida.

Registros históricos mostram que o imperador Julio Cesar utilizava uma técnica de substituição para trocar mensagens, onde o mesmo substituía a letra original da mensagem por outra, três posições à frente, hoje conhecida como Cifra de Cesar. Considerando o período histórico, onde a maioria da população era analfabeta, a Cifra de Cesar foi eficiente, método este que atualmente é decifrado em instantes.

Ao longo do tempo diversos outros métodos de criptografia foram criados e o avanço tecnológico os tornou obsoletos, por isso a criptografia moderna passou a utilizar métodos matemáticos para realizar a tarefa de cifrar, dificultando o acesso a quem não estiver autorizado.

Devido à grande facilidade de divulgação e de acesso aos dados, tem-se a necessidade da proteção dos mesmos, evitando que eles sejam acessados ou modificados por pessoas mal-intencionadas. A partir dessa necessidade de manter as informações em segredo, a criptografia entrou em cena.

Segundo Schneier (1996):

> Os textos geralmente podem ser facilmente lidos por qualquer pessoa, mas existem alguns em que a leitura não deve ser permitida a pessoas não autorizadas. Esse processo que deixa as informações de uma maneira que elas não fiquem claras, este método é chamado de criptografia e o texto com as informações ocultadas é chamado de texto cifrado.

Existem dois métodos de criptografia: a criptografia simétrica e a assimétrica. A criptografia simétrica utiliza a mesma chave para codificar e decodificar, já na criptografia assimétrica são utilizadas chaves diferentes.

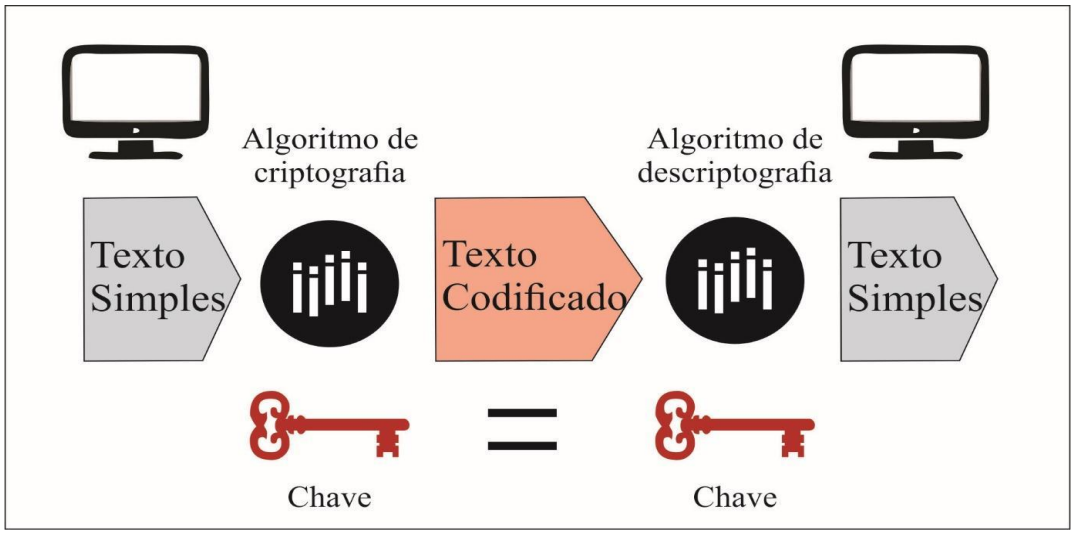
### 2.3.1 Criptografia Simétrica

É o modelo mais antigo de criptografia onde a chave - elemento que dá acesso à mensagem oculta trocada entre duas partes - é igual para ambas as partes e deve permanecer em segredo (privada).

Usualmente esta chave é representada por uma senha, usada tanto pelo remetente para codificar a mensagem numa ponta, como pelo destinatário para decodificá-la na outra (OLIVEIRA, 2006).

Segundo Ferreira (2012), "criptografia simétrica nada mais é do que um algoritmo que embaralha as informações, tornando estas ilegíveis. Para que a informação volte a ser legível, é preciso inserir a chave de sessão (*key session*) criada no processo de encriptação". O método de criptografia simétrica pode ser observado na figura a seguir.

*Figura 4: Criptografia simétrica*




Fonte: Elaborada pelos autores.

A criptografia simétrica altera o conteúdo da mensagem ou do texto de forma que apenas quem sabe qual é a chave secreta consiga ler a mensagem. Neste método de criptografia são realizados dois tipos de cifragem: por fluxo ou por bloco.

Na cifragem por fluxo a mensagem é processada *bit* a *bit*, é gerada uma sequência de *bits* que posteriormente será utilizada como chave, também chamada de *keystream.* Na cifragem por bloco, a mensagem é dividida em blocos, que geralmente são de 64 ou 128 *bits*, e cada bloco deve ser cifrado por vez de acordo com o algoritmo utilizado. Tais algoritmos realizam operações de confusão e difusão como forma de prevenção.

Difusão significa que cada *bit* do texto claro deve afetar o maior número de *bits* do criptograma (texto cifrado), este método permite que haja uma dissipação das relações estatísticas que existem entre o texto claro e o texto cifrado, desta forma estas relações ficam escondidas. Já no método da confusão, cada *bit* do criptograma deve ser uma função o mais complexa possível entre os *bits* do texto claro (SHANNON, 1949).

Dentre as vantagens que podem ser observadas na utilização da criptografia simétrica tem-se: facilidade e rapidez na execução dos processos de encriptação. Em contraposição tem-se como desvantagem a utilização da mesma chave para cifrar e para decifrar, podendo acarretar o fácil acesso à mesma, uma vez que se faz necessário compartilhar tal chave com os receptores da informação e neste processo de compartilhamento a possibilidade de interceptação da chave é grande.

O 3DES (*Data Encryption Standard)* e o AES (*Advanced Encryption Standard*) são exemplos de algoritmos que utilizam criptografia simétrica.

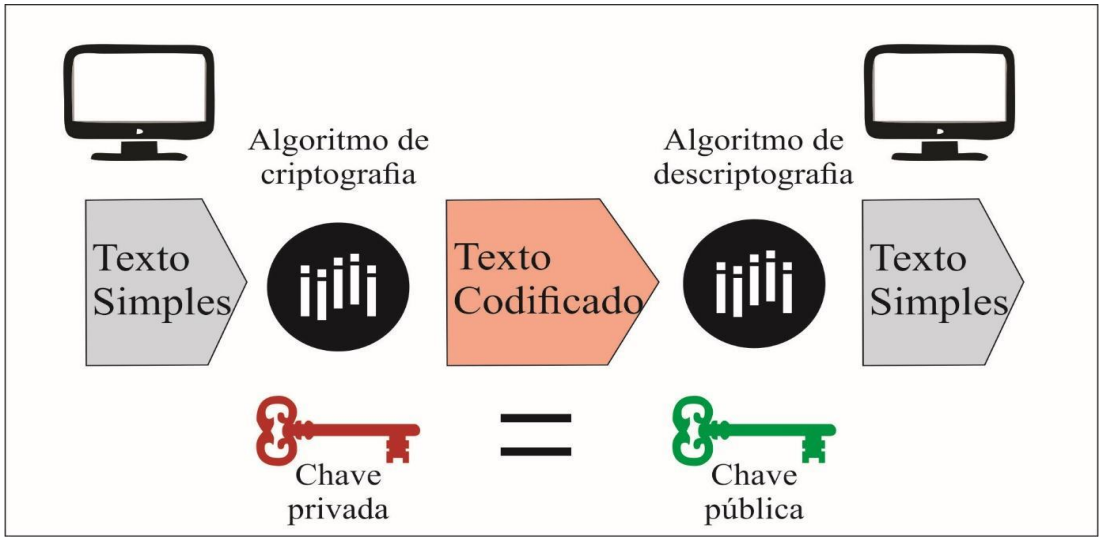
### 2.3.2 Criptografia Assimétrica

Em 1976, Diffie e Hellman mudaram os rumos da criptografia com a criptografia assimétrica. Eles propuseram um sistema para cifrar e decifrar uma mensagem com duas chaves distintas, sendo a chave pública a que pode ser divulgada e a outra mantida em segredo, denominada de chave privada.

Entre as formas de criptografia, existe também a criptografia assimétrica na qual cada parte envolvida na comunicação usa duas chaves diferentes (assimétricas) e complementares, uma privada e outra pública (OLIVEIRA, 2006).

A chave pública fica disponível à qualquer pessoa, enquanto a privada fica sob o poder de cada usuário, pois é com o uso dela que a mensagem criptografada poderá ser decodificada, como é mostrado na Figura 5.

*Figura 5: Criptografia assimétrica*




Fonte: Própria do autor.

Assim como em qualquer outro sistema, os que utilizam criptografia assimétrica devem possuir alguns requisitos, que, de acordo com Stallings (2008, p.181) são:

- Uma das duas chaves precisa permanecer secreta;
- Deverá ser impossível ou pelo menos impraticável decifrar uma mensagem se nenhuma outra informação estiver disponível;
- O conhecimento do algoritmo mais uma das chaves, mais amostras do texto cifrado precisam ser insuficientes para determinar a outra chave.

A criptografia assimétrica não está livre de ataques, e essa criptografia é considerada vulnerável ao ataque de força bruta, que por meio de tentativa e erro tenta descobrir nomes de usuários e senhas. Uma forma para tentar evitar que este ataque ocorra, é utilizando chaves grandes para que assim as senhas não possam ser facilmente descobertas.

Apesar de ser um método bastante seguro tem como desvantagem o seu tempo de processamento das mensagens em relação à utilização da criptografia simétrica.

O algoritmo de chave pública não substitui a criptografia simétrica, pois eles são lentos e vulneráveis à alguns ataques. Geralmente a criptografia assimétrica é usada para distribuir com segurança as chaves simétricas (chaves de sessão), pois esta será usada para cifrar as mensagens.

Outra vantagem da criptografia assimétrica é a permissão do envio de mensagens secretas utilizando apenas a chave pública do usuário, resolvendo assim o problema do compartilhamento da chave criptográfica como ocorre com a criptografia simétrica. A confidencialidade da mensagem é garantida enquanto a chave privada permanecer segura.

Um exemplo de um algoritmo que utiliza a criptografia assimétrica é o RSA que é considerado como uma das melhores técnicas de criptografia atualmente, esta técnica utiliza números primos, onde se multiplica dois números primos para obtenção de um terceiro número, porém a recuperação dos dois números que lhe geraram é difícil. Este método é conhecido como fatoração, este algoritmo dificulta justamente este processo de fatoração em números grandes.

## 2.4 PGP (*Pretty Good Privacy* )

Segundo Litterio (1997) "PGP é um *software* de criptografia de chave pública, altamente seguro, que realiza criptografia e descriptografia dos dados e que fornece autenticação e privacidade para a comunicação dos dados".

PGP é um sistema de codificação criado em 1991 por Phillip Zimmerman com o intuito de fazer com que o RSA funcionasse em computadores pessoais. Segundo o Guia RNP (2012), o PGP a partir de 2004 deixou de ser gratuito e isto fez com muitos usuários migrassem para o GnuPG (GPG) que é a versão livre do *software.* O GPG e PGP são compatíveis, ou seja, a partir de um dos programas é possível cifrar, decifrar, assinar e verificar assinatura entre eles.

O PGP é geralmente utilizado para criptografia e descriptografia de textos, *e-mails*, arquivos e também para assinatura digital, ele é baseado no algoritmo de criptografia IDEA (*Internacional Data Encryption Algorithm),* que possui uma estrutura semelhante ao DES.

### 2.4.1 *OpenPGP*

O *OpenPGP* é um protocolo não proprietário que implementa padrões de criptografia, assinatura digital e de certificados digitais. Foi iniciado em 1997 através de um grupo de trabalho no *Internet Engineering Task Force* (IETF). Este protocolo pode ser implementado por qualquer organização sem pagamento de licenças (Callas et al. 2007).

### 2.4.2 Codificação no *OpenPGP*

A codificação do OpenPGP se dá em duas etapas:

Na primeira etapa neste sistema, inicialmente é realizada uma compressão no texto claro, que além de economizar tempo e espaço de armazenamento, fortalece também a segurança criptográfica, pois as maiorias das técnicas utilizadas para quebrar cifras (criptoanálise) são realizadas sobre o texto claro.

Após a compressão do texto ser realizada, é criada uma chave de sessão (IDEA) que é um número aleatório gerado a partir da movimentação do *mouse*, ou das teclas que o usuário digita, a partir disso a chave é utilizada por um algoritmo de criptografia para que assim o texto claro possa ser cifrado. Após os dados estarem criptografados, a chave de sessão é criptografada utilizando a chave pública do destinatário, e então é transmitida junto com a mensagem cifrada para o receptor.

A segunda etapa inicia-se após receber a mensagem cifrada, identifica-se a chave de sessão (IDEA) cifrada e decifra-a utilizando a chave privada do receptor e somente após obter a chave de sessão que a mensagem será decifrada.

Esta ferramenta é considerada um sistema de criptografia híbrido, pois consegue utilizar tanto a velocidade do modelo de criptografia simétrica quanto o modelo que utiliza chave pública (assimétrica) sem comprometer a segurança. O *OpenPGP* pode ser utilizado como um sistema que é à prova de falsificações de assinaturas digitais e isto permite verificar a modificação ou não dos arquivos.

### 2.4.3 Certificado Digital PGP

A certificação digital, de modo geral, é uma forma de assegurar e verificar a autenticidade de um par de chaves (pública e privada). Em outras palavras, para certificar que o receptor da mensagem é de fato o verdadeiro receptor.

Como foi apresentado na seção anterior, para enviar uma mensagem codificada de um emissor A, para um receptor B, é necessário:

1: A cifra a mensagem com a chave púbica de B.

2: A envia a mensagem para B.

3: B decifra mensagem com sua chave privada.

Portanto, se A utilizar a chave pública de outra pessoa, B não será capaz de decifrar a mensagem, ou se a chave pública utilizada for de um atacante, outra pessoa obterá um acesso indevido.

Neste sentido, faz-se necessário que uma terceira parte confiável certifique a autenticidade daquela chave pública.

Certificado digital é uma informação que juntamente com uma chave pública de um determinado usuário ajuda a verificar se a chave é realmente válida, tal certificado é constituído por uma chave pública e informações sobre o usuário.

Um certificado digital PGP nada mais é do que uma chave pública com etiquetas que contém informações e assinaturas do dono da chave, uma característica deste certificado é que ele pode conter várias assinaturas.

Certificados PGP são constituídos pelas seguintes informações:

- Número de versão do PGP: versão utilizada para criar a chave associada ao certificado;
- Chave pública do detentor do certificado: parte pública do par de chaves que está associado ao algoritmo;
- Informações do detentor do certificado: contém informações sobre a identidade do utilizador;
- Assinatura numérica do detentor do certificado: assinatura efetuada com a chave privada correspondente à chave pública associada ao certificado;
- Período de validade: data de início e expiração do certificado;
- Algoritmo de codificação simétrico: algoritmo de codificação que o detentor do certificado utiliza para cifrar as informações.

## 2.5 Conclusão

Neste capítulo foram apresentados conceitos relevantes que serão utilizados no desenvolvimento do presente trabalho, tais como a arquitetura de uma nuvem computacional e seus modelos de serviço; segurança dos dados, e; como técnicas de criptografia podem evitar que os dados sejam acessados de forma indevida. Bem como foi apresentado o protocolo *OpenPGP* como alternativa de método criptográfico.

## 3 METODOLOGIA

A metodologia do trabalho que será apresentado é baseada em estudos e pesquisas bibliográficas em artigos, livros e revistas científicas visando obter um maior conhecimento acerca dos conceitos de computação em nuvem, segurança da informação, criptografia e as demais tecnologias utilizadas.

Buscou-se através da pesquisa identificar as vantagens e desvantagens da computação em nuvem bem como os riscos e perdas que a falta de segurança pode causar na sua utilização.

Primeiramente será realizado um estudo sobre os principais *frontends* utilizados por nuvens de armazenamentos de dados, suas características, vantagens e métodos de segurança presentes em suas estruturas.

Após este estudo inicial, será iniciada a preparação do ambiente de desenvolvimento, utilizando o sistema operacional Linux, do aplicativo proposto e posteriormente seu desenvolvimento de fato.

O aplicativo terá a finalidade de realizar criptografia dos dados que serão armazenados na nuvem, diretamente no cliente. Sua execução será iniciada quando for realizado o *upload* do arquivo no aplicativo onde primeiramente será requisitado o carregamento da chave que será utilizada no processo de criptografia no arquivo antes de ser enviado para o armazenamento, garantindo assim que o objetivo principal do trabalho seja alcançado.

# 4 TECNOLOGIAS DE ARMAZENAMENTO DE DADOS EM NUVEM

Neste capítulo serão apresentadas as principais tecnologias que possibilitam o armazenamento de dados em nuvem. O foco será as tecnologias de *frontend*. No entanto, também serão apresentadas algumas das principais tecnologias de *backend*.

O capítulo está dividido da seguinte forma: apresentação do cenário geral de uma nuvem de armazenamento de dados que inclui o *frontend* e o *backend*. Seguido da apresentação das principais tecnologias de *frontend*. Após isso, apresentação das principais tecnologias de *backend* suportadas pelos *frontends* apresentados. Por fim, serão apresentados os aspectos de segurança e um comparativo entre os *frontends* baseado em tais aspectos.

## 4.1 Cenário Geral de uma Nuvem de Armazenamento de Dados

Segundo apresentado no Capítulo 2, uma nuvem tem sua arquitetura dividida em: *hardware*, infraestrutura, plataforma e aplicação. Tais estruturas englobam desde recursos físicos, como servidores, até as aplicações que o usuário pode interagir. De forma a abstrair detalhes da infraestrutura que constitui a nuvem, a seguir é apresentado um cenário geral de uma nuvem de armazenamento de dados. Ela é composta pelos seguintes componentes: o cliente, canais de comunicação, o *frontend* e o *backend.*

O *frontend* abstrai o conjunto de tecnologias que interagem com os usuários. Ou seja, ele recebe as requisições dos usuários como, por exemplo, arquivos a serem armazenados e as encaminha para a nuvem. Dessa forma, o *frontend* funciona como um intermediário entre o usuário e a nuvem que efetivamente armazena os dados.

Os clientes ou usuários são os indivíduos que interagem com o *frontend* da nuvem para realização de operações como armazenamento, recuperação e alteração de arquivos. Para isso, eles podem fazer uso de aplicativos em diversas plataformas como: *desktop*, celular, *tablet* ou aplicações *Web.*

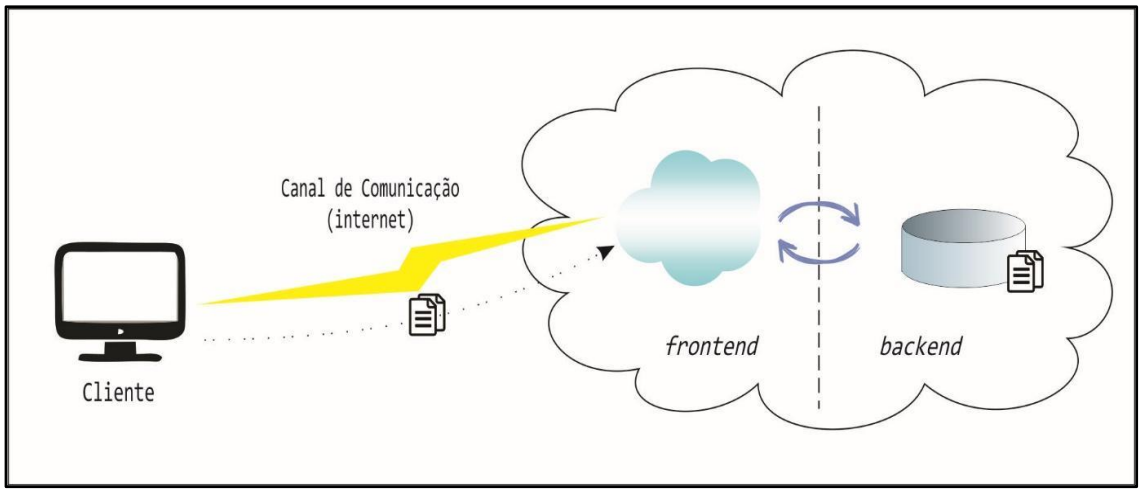
O *backend* é o componente em que o *frontend* interage diretamente, pois é nele onde efetivamente os dados dos usuários são armazenados, ou seja, as requisições dos usuários são repassadas para o *backend* para que ele as efetue, pois ele é composto por um conjunto de servidores.

Ao enviar um arquivo para o *frontend*, este arquivo é enviado do *frontend* para o *backend* que então realiza seu armazenamento na nuvem, este *backend* pode ser interno, fazer parte da mesma rede ou estrutura do *frontend* ou pode ser externo como o *Amazon Web Services*.

Em uma nuvem de armazenamento de dados também existem os canais de comunicação entre os componentes, esses canais em geral são representados pela *Internet* quando a comunicação é realizada entre os usuários e o *frontend*, no entanto, a comunicação entre o *frontend* e o *backend* também pode ser realizada dentro de outra rede específica.

A figura abaixo apresenta a visão geral de uma nuvem de armazenamento de dados. Nesta figura, o computador cliente conecta-se através da *internet* (canal de comunicação) para fazer *upload* dos seus arquivos para a nuvem, para isso ele interage com o *frontend*, este, por sua vez, se comunica com o *backend* para atender a requisição do usuário.

*Figura 6: Visão geral do armazenamento de dados em nuvem*




Fonte: Elaborada pelos autores.

## 4.2 Tecnologias de *Frontend* para o Armazenamento de Dados em Nuvem

Como apresentado, o *frontend* é o componente responsável por realizar interação com o usuário. Ele é composto por um conjunto de ferramentas que possibilitam a realização dessa interação através das mais diversas plataformas.

Embora o propósito dos *frontends* seja em geral o mesmo, possibilitar o armazenamento de arquivos, existem diferenças em suas estruturas. A seguir são apresentados os principais *frontends* utilizados na construção de nuvens de armazenamento de dados.

### 4.2.1 *Stacksync*

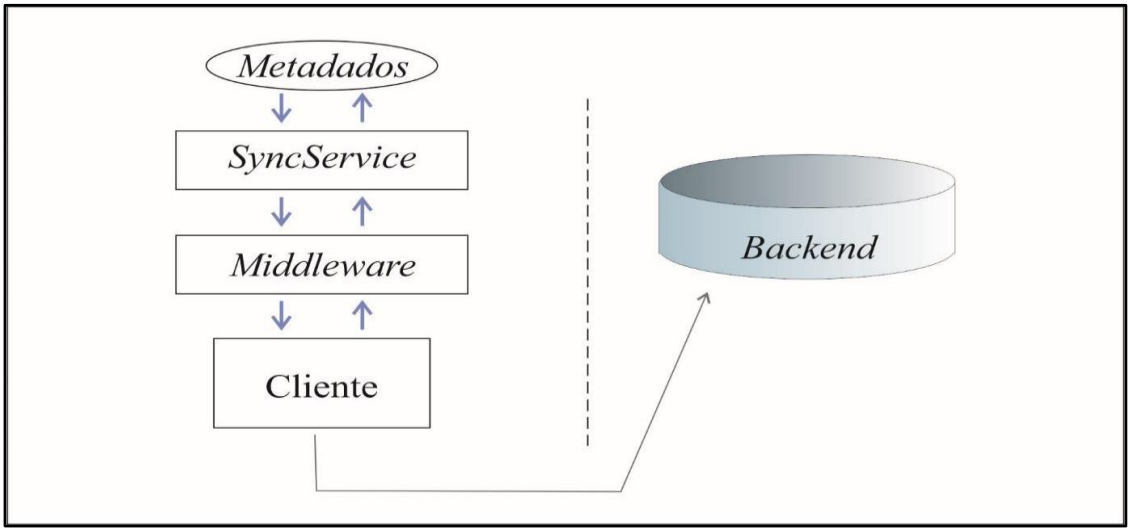
O *Stacksync*, de acordo com a sua documentação, é uma plataforma *open source* de armazenamento, sincronia e compartilhamento de dados em nuvem que trabalha integrado com o *OpenStack Swift*, é desenvolvido na linguagem Java e está disponível para: Linux, *Windows* e Mac OS, *mobile client* para Android e IOS e não possui acesso *web*. O *desktop client* do *Stacksync* é responsável por detectar alterações realizadas pelos usuários nos arquivos; responsável pelas atualizações e por registrar os erros e enviá-los ao servidor para serem corrigidos.

Esta plataforma é escalável, ou seja, projetada para atender a grande quantidade de usuários, ela permite o compartilhamento dos arquivos com um grupo de usuários e também é considerada uma solução segura de *backup* automático, pois utiliza um sistema de armazenamento com replicação que permite guardar versões diferentes dos arquivos para que os mesmos possam depois ser recuperados.

O *StackSync* é dividido em três blocos principais: os clientes, o serviço de sincronização (*SyncService*) e o *backend* de armazenamento, sua arquitetura também pode ser comparada com a arquitetura de uma aplicação *web*, pois possui três camadas.

A Figura 7 a seguir apresenta uma visão geral da arquitetura do *StackSync*, ela contém os componentes principais e a interação entre eles. O cliente *StackSync* e o *SyncService* interagem através do *middleware* de comunicação chamado *ObjectMQ*. O *SyncService* interage com o banco de dados de metadados e os clientes interagem diretamente com o armazenamento de *backend* para fazer o *upload* e *download* de arquivos.

*Figura 7: Arquitetura do StackSync*




Fonte: Elaborada pelos autores.

O armazenamento de dados do *Stacksync* dá-se da seguinte maneira: primeiramente são extraídos os metadados dos arquivos (nome do arquivo e data de modificação, por exemplo), em seguida, esses dados são armazenados em um banco de dados. Após o armazenamento dos arquivos serem realizados no *backend*, o arquivo passa a ser dividido em partes menores chamadas de ***chunks*****,** estes pedaços são unidos de volta e são utilizados quando há necessidade da reconstrução do arquivo original.

Esta plataforma separa metadados dos fluxos de dados armazenados dividindo o serviço de *backend* em dois componentes separados: a parte que hospeda os arquivos do usuário e os objetos que é chamada de *backend* de armazenamento e a outra parte que é responsável pela gestão do arquivo de sincronização dos metadados, chamado de *backend* de metadados. O gerenciamento dos metadados associados aos arquivos inclui versionamento, histórico de modificação de atributos e tempo da última modificação.

O *StackSync* utiliza o *PostgreSQL* que é responsável por armazenar informações dos arquivos de metadados e o *OpenStack Swift* como *backend* de armazenamento sendo possível também utilizar FTP (*File Transfer Protocol*).

O *StackSync* possui métodos de encriptação tanto no cliente quanto no servidor. No cliente, a chave de criptografia dos arquivos é de propriedade exclusiva do usuário, quando a criptografia dos arquivos é realizada no servidor, o próprio servidor é responsável pela segurança dessa chave. A versão que realiza criptografia de dados no servidor está disponível para instalação em nuvens híbridas e privadas.

A seguir são apresentadas as principais funcionalidades do *StackSync*:

- Possibilita criptografia de dados tanto no cliente quanto no servidor;
- Alta escalabilidade da infraestrutura;
- Versionamento de arquivos;
- Compartilhamento de arquivos;
- Realiza cópia dos dados para aplicações realizadas *offline* e que ainda não foram sincronizadas;
- Possibilita integração entre os sistemas operacionais;
- Interoperabilidade;
- Permite compartilhamento com grupo de usuários.

Como método de ciframento dos dados, o *StackSync* utiliza a criptografia AES-256 bits, que é responsável por evitar que terceiros tenham acesso à informação e para transmissão dos dados de forma segura entre cliente e servidor, é utilizado o protocolo SSL.

O cliente possui uma base local de dados e uma *thread* que monitora o estado em que se encontram as pastas que foram sincronizadas, para evitar possíveis conflitos com operações que foram realizadas *offline*, o *StackSync* oferece algumas políticas similares ao *Dropbox*, onde uma cópia do documento que está em conflito é criada e então o usuário decide qual ação será realizada com o arquivo.

O serviço de sincronização, também conhecido como *SyncService*, é a parte mais importante da arquitetura de sincronização pois ele é responsável pela gestão de metadados. Quando os clientes *desktop* comunicam-se com o *SyncService* é possível conseguir acessar as alterações que ocorreram quando eles estavam *offline* e novas versões de um arquivo podem ser submetidas.

### 4.2.2 *OwnCloud*

O *OwnCloud*, segundo o manual *OwnCloud* 9.0 *user manual*, é uma ferramenta de *frontend* para nuvem de armazenamento de dados, de código aberto e que dá ênfase à criptografia de dados no servidor e está disponível para os sistemas operacionais: *Windows*, Mac OS, Linux, Android e iOS.

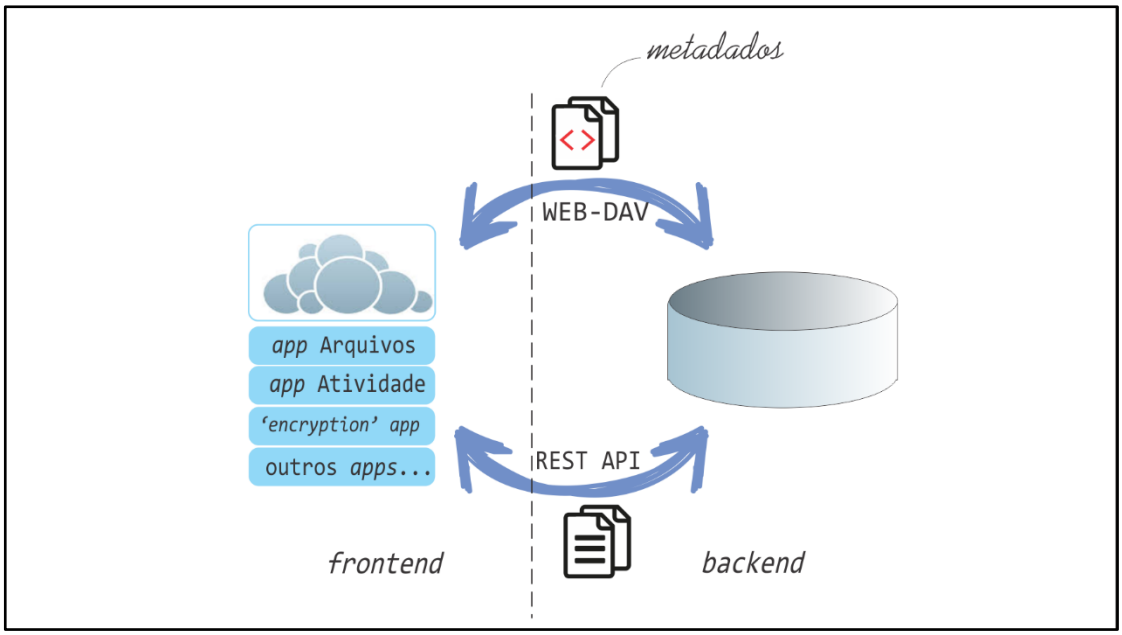
Quando sua primeira versão foi lançada, a ferramenta tinha como proposta sincronizar arquivos entre vários dispositivos, característica útil quando se compartilha arquivos entre vários usuários ou apenas para realização de *backup*. No entanto, o projeto cresceu e tornou-se um poderoso *frontend*, altamente escalável propriedade essa que é pode ser observada devido a existência de vários *apps* que implementam diversas funcionalidades.

Quando as informações são compartilhadas com o *OwnCloud*, elas são imediatamente sincronizadas com o *backend* de armazenamento que é o ponto central do armazenamento de dados sendo responsável pelo envio de novos arquivos para os clientes. O cliente *OwnCloud* é a parte responsável por decidir quando os dados devem ser enviados e por fazer *download* dos arquivos que estão armazenados no servidor.

Esta ferramenta possibilita o armazenamento e sincronização de arquivos sem limite de espaço de armazenagem (com exceção da capacidade do disco rígido) ou o número de clientes conectados. Os arquivos e dados armazenados podem facilmente ser acessados utilizando a interface *web* ou utilizando o aplicativo cliente de sincronização.

O *ownCloud* utiliza o MySQL que é responsável pelo armazenamento dos arquivos e ele também é compatível com diversos *backends*, como por exemplo, o *OpenStack Swift*. Também é possível utilizar armazenamento externo, tal como *Google Drive*, *Dropbox*, *Amazon S3* e FTP.

*Figura 8: Visão geral do OwnCloud*




Fonte: Elaborada pelos autores

A plataforma permite a criação de grupos de usuários de compartilhamento de arquivos e pastas, onde se pode compartilhar um arquivo com usuários *ownCloud* ou com usuários externos através da criação de um *link* para acesso onde há a possibilidade de filtrar quais arquivos e pastas estão compartilhados e quais o usuário compartilhou.

A seguir encontra-se uma lista com as principais funcionalidades do *ownCloud*:

- Armazenamento de arquivos;
- Sincronização de clientes;
- Possui editor *online* de texto com analisador de sintaxe;
- Permite salvar os “favoritos” assim como em outros navegadores;
- Possui galeria de fotos;
- Permite visualizar arquivos em formato PDF;
- Possui organizador de tarefas;
- Criptografia de dados;
- Permite criação de *apps* personalizados para executar as mais variadas tarefas.

Esta plataforma permite a criptografia dos dados armazenados no servidor, para isto é necessário habilitar o aplicativo referente a esta funcionalidade. Quando a criptografia é ativada, todos os arquivos, independentemente de estarem armazenados no servidor do *OwnCloud* ou em armazenamentos externos, serão criptografados e o acesso a estes dados criptografados só pode ser realizado através do próprio *OwnCloud*, que é responsável pela gerência das chaves criptográficas e, portanto, realiza a criptografia e a descriptografia deles.

O aplicativo responsável por fazer a criptografia é chamado de *Encryption App*, ele tem como finalidade principal criptografar arquivos que serão armazenados em serviços externos. Apenas os usuários que possuem as chaves criptográficas têm acesso ao conteúdo das pastas e aos arquivos criptografados aqueles que não possuem essa chave conseguirão apenas visualizar as pastas e os nomes dos arquivos.

O O*wnCloud* oferece controle de versão dos arquivos, onde cópias de segurança são criadas possibilitando que os arquivos possam voltar a versões anteriores. As versões antigas expiram automaticamente para garantir que o usuário não corra risco de ficar sem espaço disponível em disco.

### 4.2.3 *Pydio* (*Put Your Data in Orbit)*

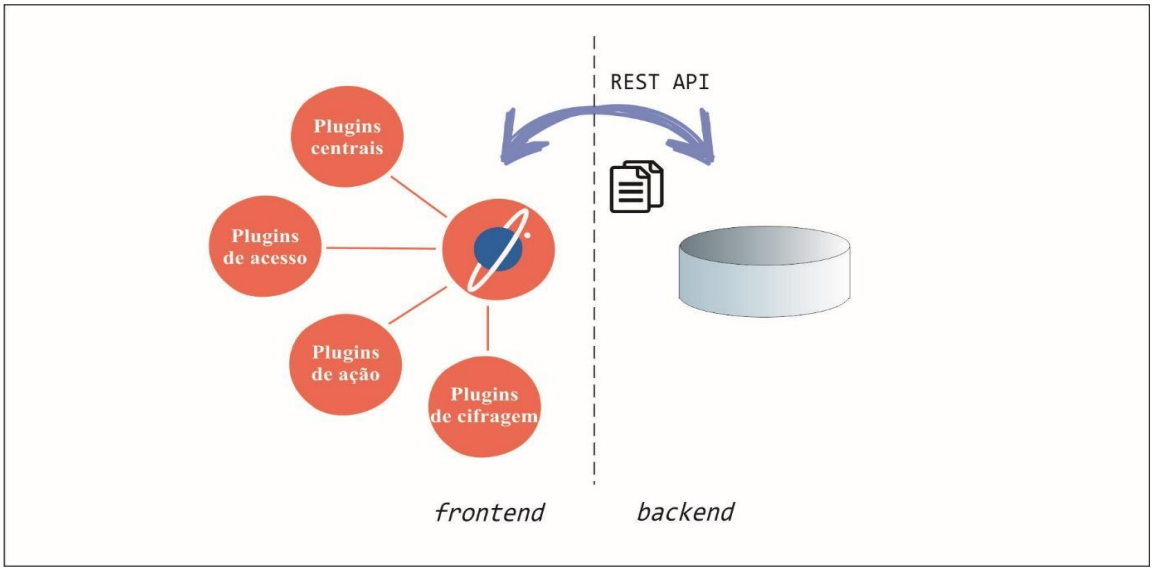
O *Pydio* é uma plataforma *open-source* de armazenamento e de sincronização de dados em nuvem. Ela permite que dados sejam armazenados em um *backend* de armazenamento e depois sejam sincronizados com um cliente.

Esta plataforma pode ser utilizada tanto através de um navegador por interface *web* quanto por uma aplicação cliente denominado *PydioSync*, compatível com *Windows*, Mac e Linux.

Além de ser utilizado para sincronização, o *Pydio* também pode ser utilizado apenas como um centro de armazenamento de dados. O armazenamento desses arquivos é feito através de um servidor de *backend* e é realizado anteriormente à sincronização com computadores ou dispositivos móveis

O *Pydio* é baseado em *plugins,* como apresentado na Figura 9. Esses *plugins* são responsáveis por todas as funcionalidades suportadas pela plataforma, desde o *login,* o *upload* de dados até a comunicação com diferentes *backends* que são utilizados juntamente com o *WEB-DAV,* como por exemplo HP *Cloud, Amazon S3,* Samba e FTP.

*Figura 9: Visão geral do Pydio*




Fonte: Elaborada pelos autores

*Pydio* utiliza tanto o MySQL quanto o MariaDB como banco de dados, dependendo da distribuição Linux escolhida para ser utilizada.

Esta plataforma possui características que permitem compartilhar arquivos e pastas tanto com os usuários internos quanto com os externos e também permite a criação de *sites* para a publicação de documentos na *Internet.*

Todas estas funcionalidades são providas pelos *plugins* do *Pydio*, conforme apresentado anteriormente. A seguir são listadas as principais características do Pydio:

- O acesso e compartilhamento dos dados podem ser feitos de qualquer dispositivo;
- Permite versionamento e restauração dos arquivos;
- Permite a utilização de grande volume de dados para conteúdo de pesquisa;
- Permite recuperação de arquivos;
- Pré-visualização integrada para os formatos mais comuns de arquivos;
- *Links* compartilhados podem ser protegidos por senha;
- Auto-expiração dos *links* após um determinado número de *downloads*;
- Visualização, edição ou compartilhamento de arquivos com apenas um clique;
- Permite definir por quantos dias o arquivo ficará disponível;
- Define quantos downloads serão permitidos;
- Define uma senha para que os arquivos possam ser acessados;
- Envia notificação quando um arquivo é visualizado ou editado;

- Imagens podem ser editadas diretamente no navegador com a utilização do PIXLR editor;
- Documentos do *Office* podem ser editados dentro do *Pydio* com o *Zoho* ou com o *Libre Office*;
- Em dispositivos móveis, os arquivos podem ser compartilhados utilizando a aplicação nativa de *e-mail* do celular.

Assim como no *OwnCloud*, a comunidade pode desenvolver e disponibilizar novos *plugins*.

A sincronização realizada no *desktop* permite a criação de uma pasta local para sincronizar com a *interface web*, este aplicativo de sincronização, como mencionado anteriormente, é chamado de *PydioSync*. O *PydioSync* é multiplataforma e suas tecnologias são criadas para lidar com grande quantidade de dados, o aplicativo também conta com atualizações regulares das suas funcionalidades.

Algumas características do *PydioSync*:

- Permite definir os intervalos de sincronização;
- Pode ser configurado apenas para *upload*, apenas para *download*, ou para ambos;
- Apenas uma subpasta de um *workspace* pode ser escolhida para ser sincronizada;
- O administrador pode definir e limitar os parâmetros de sincronização.

Nas aplicações do *Pydio* para dispositivos móveis é possível abrir qualquer tipo de arquivo, eles podem ser editados utilizando aplicações externas e seu *upload* pode ser facilmente realizado após as modificações terem sido feitas. Arquivos que estão armazenados localmente no dispositivo são sincronizados novamente sempre que o dispositivo se conectar a *internet.*

Para criptografia dos arquivos ele utiliza o *plugin* ENCFS *mount* que permite que os usuários cifrem os dados armazenados no servidor. Atualmente é utilizada a ENCFS (*Encryption File System*) que se baseia na criptografia dos dados através do sistema de arquivos. A decodificação dos arquivos criptografados só poderá ser feita durante uma sessão de usuário.

Recomenda-se acessar o *Pydio* utilizando o protocolo HTTPS para evitar a interceptação de qualquer informação durante a comunicação entre o ele e o navegador, uma vez que o protocolo HTTPS utiliza um canal criptografado.

### 4.3 Tecnologias de *backend*

Como apresentado, o *backend* é o componente que efetivamente armazena os dados na nuvem após receberem as requisições do *frontend*. A seguir são apresentadas as três principais tecnologias de *backend* utilizadas no *OwnCloud*, *Pydio* e *Stacksync*.

#### 4.3.1 *OpenStack Swift*

O *Openstack* é uma plataforma de código aberto para computação em nuvem, tanto privada quanto pública, essa ferramenta possui módulos para o armazenamento de dados, um destes módulos é o *OpenStack Swift.*

O *OpenStack Swift* é o software responsável por garantir a replicação e a integridade dos dados no cluster, estes dados podem ser modificados e recuperados sempre que for necessário e a redundância de dados (cópia idêntica replicada) pode ser configurada durante o processo de instalação.

De acordo com Biswas et al. (2014), a plataforma *OpenStack Swift* compreende um sistema de armazenamento de objetos de código aberto e que permite a construção de *clusters* de armazenamento, para combinar o desempenho e a alta disponibilidade em sistemas distribuídos.

*Swift* é um sistema de armazenamento de objetos amplamente utilizado, popular e *open source*, ele é projetado para armazenar arquivos, vídeos, conteúdo *web*, backups e imagens, por exemplo.

#### 4.3.2 FTP (*File Transfer Protocol)*

O FTP é um protocolo de transferência de arquivos responsável pelo envio dos arquivos para *web.*

Segundo Postel e Reynolds (1985):

> Os objetivos do FTP são: promover o compartilhamento de arquivos ou programas e incentivar indiretamente uso de computadores remotos para proteger o usuário de variações nos sistemas de armazenamento de arquivos entre hosts.

A transferência dos arquivos é feita entre um servidor e um cliente. No servidor, os dados ficam hospedados e no cliente as operações são realizadas. A conexão ao servidor FTP requer autenticação através de usuário e senha, no entanto, pode-se habilitar conexões anônimas.

O servidor FTP é um computador central que é responsável pelo gerenciamento e fornecimento dos dados aos computadores que estão conectados a ele, é também denominado como *host.* O cliente FTP pode ser qualquer computador que esteja acessando ou utilizando os recursos de um servidor que esteja em uma rede local ou na *internet.*

### 4.3.3 Amazon S3

O *Amazon* S3 é um serviço de armazenamento de dados oferecido pela *Amazon*. Os dados dos usuários são armazenados na nuvem da Amazon, onde seu uso é controlado para efeitos de cobrança.

De acordo com o que pode ser observado na descrição do site da *Amazon Web Services*, o *Amazon* S3 é um serviço armazenamento de arquivos que possui uma *interface* de *web service* simples para armazenamento e recuperação de arquivos. A opção de migração de dados para a nuvem permite que grandes volumes de dados sejam transferidos e depois que os dados forem armazenados no S3, eles poderão ser automaticamente divididos em categorias de armazenamento.

Este serviço de armazenamento de arquivos considerado altamente escalável, de alta velocidade e de baixo custo, onde paga-se apenas pelo armazenamento que estiver sendo utilizado, permite que seus usuários realizem *uploads*, *downloads* e armazenamentos de arquivos com tamanho de até 5GB. Sua interface, baseada em *Web*, utiliza criptografia para realizar autenticação do usuário.

Os usuários deste serviço possuem a opção de escolha entre manter os dados privados ou torná-los acessível ao público e também possuem a opção de criptografar os dados antes de serem armazenados.

## 4.4 Comparativo entre os *frontends* de armazenamento em nuvem

De acordo com o apresentado na seção 3.2, o *Pydio*, o *StackSync* e o *OwnCloud* são plataformas de código aberto que são utilizados para armazenar dados em uma nuvem de armazenamento, sendo possível sincronizá-los com um computador pessoal ou dispositivos móveis. No entanto, essas ferramentas possuem diferenças entre elas, seja na forma de sincronização, compartilhamento ou até mesmo em formas de acesso.

As seções seguintes apresentam um comparativo das soluções de *frontend* abordadas anteriormente em relação a algumas funcionalidades gerais, como armazenamento e sincronização dos dados; compartilhamento, colaboração e versionamento de arquivos, respectivamente.

Também são abordados aspectos de segurança, uma vez que as nuvens de armazenamento de dados devem ter como um de seus fundamentos a segurança de dados. No entanto, a segurança é muitas vezes deixada em segundo plano nas ferramentas que compõem a nuvem. (MOIA; HENRIQUES, 2014)

### 4.4.1 Armazenamento e sincronização dos dados

Nesta seção é demonstrado como o armazenamento e a sincronização dos dados é realizada em cada tecnologia estudada. Entende-se como armazenamento o ato de gravar os dados enviados pelo usuário para a nuvem para serem guardados. E como sincronização, o ato de manter uma cópia de uma pasta local no computador do usuário igual a uma pasta localizada na nuvem de armazenamento de dados.

Em relação ao armazenamento, tanto o *Pydio*, quanto o *Stacksync* e o *OwnCloud* não só permitem armazenar os dados localmente nos servidores da nuvem, ou seja, *frontend* e *backend* no mesmo servidor, mas também permitem adicionar armazenamento externo como: FTP, *Swift* e *Amazon* S3, por exemplo.

Para a sincronização entre servidor e o cliente o *Pydio* utiliza o recurso *PydioSync*, que permite controlar a maneira com que os arquivos serão armazenados em tempo real, manualmente ou baseado em tempo, os administradores podem controlar o que os usuários poderão compartilhar além disso, o processo de sincronização de arquivos pode ser pausado e retomado posteriormente. O *OwnCloud* utiliza *OwnCloud Desktop Client* para sincronizar os arquivos do *desktop* para o servidor *OwnCloud,* o arquivo precisa apenas ser copiado para a pasta e o cliente é responsável por realizar a sincronização. Já no *Stacksync* o responsável é o *Stacksync Client*, um aplicativo que monitora as pastas locais e sincroniza-as com um repositório de *backend.*

Além dos aplicativos cliente citados no parágrafo anterior, os arquivos e dados armazenados na nuvem podem ser facilmente acessados utilizando a *interface web*, no *OwnCloud* e no *Pydio*. O *OwnCloud* também permite sincronizar uma ou mais pastas, através do recurso de controle de versão, onde a versão anterior dos arquivos pode ser acessada e os arquivos que foram apagados também podem ser recuperados. No *StackSync* o acesso é feito apenas via cliente, não possuindo uma interface *web* específica para realizar esta operação.

### 4.4.2 Compartilhamento e colaboração

O compartilhamento de arquivos é uma funcionalidade básica em nuvens de armazenamento de dados, pois os usuários a utilizam para distribuir arquivos que estão salvos e até mesmo criar espaços virtuais de trabalho isto porque as nuvens oferecem grande flexibilidade quando se refere à disponibilidade da informação.

Em relação a compartilhamento e colaboração o *Pydio* possui características que permitem compartilhar arquivos e pastas tanto com os usuários internos quanto com os externos, assim como a criação de *sites* para a publicação de documentos na *internet*. O *OwnCloud* permite a criação de grupos de usuários de compartilhamento de arquivos e pastas, onde pode-se compartilhar um arquivo com outros usuários *OwnCloud* ou com o público externo através da criação de um *link*. Na plataforma também há a possibilidade de filtrar quais arquivos e pastas estão compartilhados com o usuário e quais ele compartilhou.

### 4.4.3 Versionamento de arquivos

O versionamento dos arquivos nada mais é que manter versões dos dados após cada alteração sofrida quando o mesmo já está armazenado na nuvem.

O *OwnCloud* suporta um sistema considerado simples para realizar o controle de versão dos arquivos. Este versionamento cria cópias de segurança dos arquivos que estão armazenados e há a possibilidade de reversão para um arquivo anterior, se as versões armazenadas excederem 50% do espaço livre disponível, o *OwnCloud* exclui as versões anteriores para aumentar a disponibilidade de espaço.

No *OwnCloud* as versões antigas ficam guardadas por diferentes períodos de tempo, por exemplo, na primeira hora são mantidas a cada minuto, em 24 horas é mantida uma versão a cada uma hora, já no período de 30 dias as versões são mantidas diariamente, após esse período, as versões são mantidas semanalmente. Os arquivos deletados são mantidos por um período de 180 dias.

A funcionalidade de versionamento no Pydio permite que seja mantido um controle nas alterações que forem feitas e atualmente baseia-se no GIT (sistema de controle de versão de arquivos), sistema esse que é aconselhável na utilização para arquivos de mídia pois ele não aceita armazenamento de arquivos muito grandes, neste caso, aplicações externas precisarão ser utilizadas para que estas versões maiores sejam armazenadas. O usuário é capaz de ver uma lista de ações do que foi realizado em um determinado arquivo, estas versões anteriores podem ser baixadas ou pode-se apenas reverter a versão atual para uma versão anterior.

O *StackSync* utiliza um sistema de armazenamento com replicação tripla e permite guardar versões diferentes dos arquivos para que os mesmos possam depois ser recuperados, ou seja, caso um arquivo seja excluído por engano pode-se restaurar para uma versão anterior, desta forma a plataforma é considerada uma solução segura de backup automático.

### 4.4.4 Criptografia de dados no servidor

Com a finalidade de mitigar ameaças relacionadas à confidencialidade dos dados armazenados na nuvem, a criptografia de dados no servidor é utilizada para evitar que outros usuários ou atacantes obtenham acesso indevido a alguma informação. Neste tipo de criptografia as chaves utilizadas são geradas a partir da conta de usuário e senha de *login* e ficam armazenadas no próprio servidor.

O servidor consegue criptografar os dados depois de fazer o *upload* e decifrá-los antes de fazer o *download* novamente de forma transparente para o usuário.

Até a data do estudo sobre o *StackSync*, esta funcionalidade não estava disponível, isto porque o *frontend* adota a criptografia apenas no cliente.

O *Pydio* utiliza a ENCFS que é uma tecnologia de criptografia baseada em sistemas de arquivos que também precisa ser instalada no servidor de armazenamento da nuvem.

No *ownCloud* esta funcionalidade é realizada através de um aplicativo de criptografia existente em sua estrutura, o *Encryption App*, este *app* deve ser ativado pelo administrador da nuvem. As chaves criptográficas são diferentes para cada usuário.

### 4.4.5 Criptografia de dados no cliente

A criptografia de dados no cliente fornece ao usuário mais segurança aos seus dados quando comparado à criptografia de dados no servidor, isto porque o provedor não obtém nenhuma informação sobre o arquivo que está sendo armazenado e as chaves criptográficas ficam sob responsabilidade do usuário evitando assim que os ataques feitos ao servidor sejam eficientes em obter os arquivos que ali estejam armazenados.

Antes de serem enviados para a nuvem, os dados são primeiramente criptografados no computador do usuário utilizando uma chave secreta determinada por ele próprio. Somente após isso, os dados são enviados para a nuvem. Como resultado, não é possível que terceiros (e.g. administradores dos servidores) acessem o conteúdo dos arquivos sem a autorização do usuário. Ressalta-se, no entanto, que o cliente é responsável pela geração e pelo armazenamento das chaves criptográficas.

Dentre os *frontends* estudados, somente o *Stacksync* possui tal funcionalidade. Os arquivos são criptografados no aplicativo *Desktop Client* antes de serem enviados para armazenamento na nuvem, e as chaves criptográficas ficam sob a posse do cliente; na versão *Mobile Client* esta funcionalidade ainda não está disponível.

O presente trabalho desenvolve um *app* para o *OwnCloud* que visa atender a tal aspecto de segurança, ou seja, ele realiza a criptografia dos dados no cliente. No entanto, este *app* é desenvolvido exclusivamente para o cliente *web*, conforme é apresentado no capítulo 4.

### 4.4.6 Criptografia do nome dos dados

A criptografia do nome dos dados objetiva ocultar quaisquer informações sobre o arquivo salvo na nuvem, pois o nome do arquivo pode revelar informações importantes sobre o seu conteúdo.

Quando o nome do arquivo revela informações sobre o conteúdo torna-se um problema, uma solução seria a implantação de criptografia no nome do arquivo de forma que para o usuário proprietário é um processo transparente**.**

Moia e Henriques (2014) dizem:

> Assim, o nome do arquivo seria cifrado com uma chave secreta e isso evitaria que o nome revelasse algo sobre o arquivo e até mesmo o problema de se saber que o arquivo foi particionado. A chave secreta pode ou não ser a mesma usada na criptografia dos dados, dependendo dos procedimentos de gerência de chaves adotados.

Esta funcionalidade não foi encontrada em nenhuma das tecnologias de *frontend* estudadas.

### 4.4.6 Armazenamento de *logs*

*Log* é o registro de atividade gerado por programas e serviços de um computador, ele pode ficar armazenado em arquivos, na memória do computador ou em bancos de dados. Eles são essenciais para notificação de incidentes, pois permitem que diversas informações importantes sejam detectadas, como por exemplo: a data e o horário em que uma determinada atividade ocorreu, o endereço IP de origem da atividade, as portas envolvidas e o protocolo utilizado no ataque, segundo a Cartilha de Segurança para *Internet* do *Cert.br*.

Os *logs* de sistema devem ser armazenados para fins de auditoria, necessário para segurança de arquitetura e de sistemas, eles são necessários quando há a ocorrência de incidentes, tais como: parada de servidor ou até mesmo alteração em banco de dados.

Estes registros possibilitam o acompanhamento do que acontece em um ambiente informatizado. Por isso é importante que eles sejam monitorados com frequência para possibilitar que eventuais problemas sejam identificados e solucionados. Neste sentido, o gerenciamento de *logs* envolve um amplo conjunto de atividades, como foi proposto por Muller (2014):

- Análise Léxica e Transformação: processo que analisa as linhas de mensagens de *log* e produz uma saída formatada em um padrão mais adequado para futuro processamento;
- Transmissão e Recepção: processo que transmite os eventos de *log* para um servidor central em conjunto com o processo que recebe as mensagens de *log* no servidor;
- Análise Sobre Eventos de *Log*: processo, que por meio de algoritmos, regras ou consultas, extrai as informações relevantes sobre mensagens de *logs*;
- Compactação: processo que reduz o tamanho de uma informação por meio de algoritmos de compactação de texto;
- Armazenamento: processo que compreende guardar os registros de *logs* por um tempo determinado;
- Indexação e Busca: processo responsável por estruturar os registros de *logs* de forma que sejam posteriormente pesquisados;
- Visualização: processo que permite a visualização dos registros de *logs* sejam eles em tempo real ou em registros já armazenados.

Segundo Muller (2014):

> Devido ao fato de armazenar dados gerados pelo sistema, aplicações, rede, atividades dos usuários, entre outros, os *logs* fornecem inúmeras informações que se transformam em indicadores capazes de medir os níveis de segurança e de avaliar se medidas de segurança estão surtindo o efeito esperado e planejado. A melhor forma de verificar a extensão de um incidente de segurança, identificando que ativo foi violado e que a informação foi exposta é por meio dos registros de logs.

O *OwnCloud* possui alguns parâmetros de níveis de *log* que é configurado pelo administrador da nuvem, são eles: 0 - *DEBUG*, registra detalhadamente todas as atividades; 1 - *INFO*, registra atividades como *logins* de usuários e atividade de arquivos, além de erros fatais; 2 - *ERROR*, guarda *logs* de operações que falharam, e; 3 - *FATAL*, registra os logs de quando o servidor para. Por padrão é criado um arquivo de log chamado owncloud.log com o parâmetro nível 2. Já no *Pydio*, o nível padrão do log é 1. No *Stacksync*, os arquivos de log são gerados apenas quando há a existência de erros, os mesmos são enviados à um servidor onde serão armazenados para posteriormente serem analisados e corrigidos. O objetivo do servidor de log do *StackSync* é receber logs de erro dos clientes e armazená-los em banco de dados para depois serem revisados e então ser feito o estudo dos erros.

### 4.4.7 Resumo Comparativo

Pode-se observar então, que as tecnologias *open source* que foram apresentadas, seguem a mesma tendência: mesmas tecnologias de *backend*, semelhantes formas de administração do *frontend* e do aplicativo cliente, reforçando assim a propensão mundial de migração de uso de armazenamento local de arquivos para o armazenamento em nuvem.

No quadro a seguir, pode-se observar um resumo das funcionalidades abordadas nas seções anteriores.

*Quadro 1 - Resumo comparativo das ferramentas de frontend*

| | **Soluções** | | |
|---|---|---|---|
| **Aspectos de Segurança** | ***OwnCloud*** | ***Pydio*** | ***StackSync*** |
| Permite sincronização com pasta local | ● | ● | ● |
| Acesso via navegador de *internet* | ● | ● | - |
| Compartilhamento de arquivos entre usuários | ● | ● | ● |
| Compartilhamento de arquivos com usuários externos | ● | ● | - |
| Criptografia de dados no servidor | ● | ● | - |
| Criptografia de dados no cliente | - | - | ● |
| Criptografia do nome dos dados | - | - | - |
| Armazenamento de logs | ● | ● | ● |
| Versionamento dos dados armazenados na nuvem | ● | ● | ● |
| Facilidade para desenvolvimento de apps. | ● | - | - |
| Presença de métodos criptográficos em sua estrutura sem a necessidade da utilização de tecnologias externas | ● | - | - |

Fonte: Elaborada pelos autores

## 4.5 O Desenvolvimento de Apps no OwnCloud

Conforme explicado na seção 4.2.2 o *OwnCloud* é uma plataforma *open source* altamente escalável, graças a sua estrutura baseada em *apps*. Devido essa possibilidade de livre desenvolvimento de *apps*, esta plataforma foi escolhida para o desenvolvimento da parte pratica deste trabalho, cuja descrição completa é abordada no próximo capítulo.

O *OwnCloud* é composto por *apps*, como por exemplo o *app* "Arquivos", que permite o armazenamento dos arquivos; o *app* "Atividade", que registra todas as interações do usuário com a plataforma e o *Encryption App*, que provê a criptografia dos dados no servidor.

Nesta seção é demonstrado como criar um *app* para o *OwnCloud.*

### 4.5.1 Pré-requisitos

Para fins de demonstração, considera-se a instalação do *OwnCloud* em um servidor Linux com Ubuntu versão 16.04, de acordo como foi demonstrado na seção 4.2.2.

Para a criação de um *app* é necessário instalar a ferramenta de desenvolvimento do *OwnCloud*: OCDEV, uma biblioteca do *Python* 3, de acordo com o passo a passo a seguir.

- **Instalação da ferramenta de desenvolvimento do *OwnCloud* OCDEV**

Para instalar a ferramenta de desenvolvimento do OwnCloud, é necessário instalar o gerenciador de pacotes do Python 3, PIP. Para isso basta executar os seguintes comando no terminal:

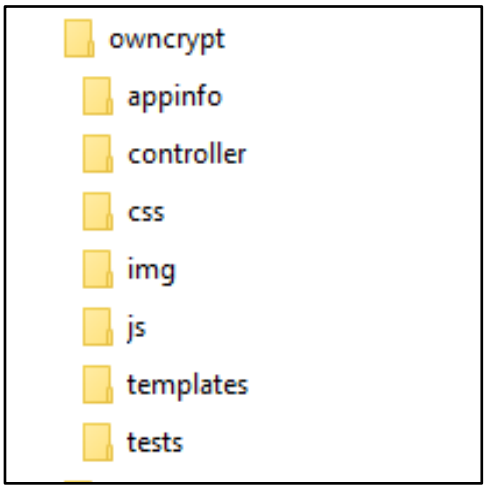
```
sudo apt-get install python3 python3-pip
sudo pip3 install ocdev
```

### 4.5.2 Criando um app

A ferramenta de desenvolvimento cria a estrutura de pastas e arquivos necessários para o desenvolvimento do *app* através da seguinte linha de comando:

```
ocdev startapp OwnCrypt
```

*Figura 10: Estrutura padrão de pastas em um app do OwnCloud.*




Fonte: Elaborada pelos autores

Uma vez criada a estrutura padrão de pastas, deve-se alterar os arquivos corretos para a realização do que se pretende fazer.

### 4.5.3 Arquivos fundamentais de um *app* no *OwnCloud*

A seguir, uma explicação sobre os principais arquivos em um *app* para o *OwnCloud*.

- **appinfo/routes.php**

Deve-se utilizar este arquivo para conectar uma URL a um método de controle. A configuração deve ser passada através de um *array* para o método ***registerRoutes***. Um exemplo de configuração do ***routes.php*** pode ser observado na figura a seguir:

*Figura 11: Exemplo de arquivo routes.php.*

```php
<?php
namespace OCA\MyApp\AppInfo;

$application = new Application();
$application->registerRoutes($this, array(
    'routes' => array(
        array('name' => 'page#index', 'url' => '/', 'verb' => 'GET'),
    )
));
```

Fonte: Manual do Desenvolvedor do *OwnCloud*.

Os métodos de controle estão configurados no arquivo ***pagecontroller.php*** na pasta *controller*, cuja descrição está a seguir.

- **controller/pagecontroller.php**

Neste arquivo são configurados os métodos de controle que foram configurados no arquivo ***routes.php***. No caso do *OwnCrypt*, eles retornarão as páginas utilizadas no app, como pode-se observar na figura a seguir.

*Figura 12: Exemplo de arquivo pagecontroller.php*

```php
<?php
namespace OCA\OwnCrypt\Controller;

use OCP\IRequest;
use OCP\AppFramework\Http\TemplateResponse;
use OCP\AppFramework\Http\DataResponse;
use OCP\AppFramework\Controller;

class PageController extends Controller {

    private $userId;

    public function __construct($AppName, IRequest $request, $UserId){
        parent::__construct($AppName, $request);
        $this->userId = $UserId;
    }

    public function index() {
        $params = ['user' => $this->userId];
        return new TemplateResponse('owncrypt', 'main', $params);  // templates/main.php
    }
}
```

Fonte: Manual do Desenvolvedor do *OwnCloud*.

- **js/script.js**

Neste arquivo está contido os scripts que são executados na página do *app*. Este é um arquivo *JavaScript* padrão, outros podem ser necessários para o desenvolvimento do *app* de acordo com a finalidade.

Os scripts devem ser importados dentro dos *controllers*, através da função **/OCP/Util::addscript('app', 'script')**. Esta função faz parte da API do *OwnCloud*, apresentada a seguir.

### 4.5.4 API *OwnCloud*

Uma API (*Application Programming Interface*) é um conjunto particular de regras que podem ser utilizados para realizar uma tarefa e/ou comunicar programas.

O *OwnCloud* possui uma API própria que automatiza várias funções na plataforma, tais como mover um arquivo no servidor, criar novos arquivos e diretórios, gerar URL etc. A lista completa dos métodos implementados na API do *OwnCloud* pode ser consultada no site *api.owncloud.com.*

No capítulo 5 é apresentado o *app* desenvolvido e as funções utilizadas para seu funcionamento.

## 4.6 Conclusão

Neste capítulo foi apresentado como é dividida a arquitetura de uma nuvem e que além de sua infraestrutura e aplicações ela também é composta por canais de comunicação, pelo *frontend* e pelo *backend*.

*StackSync, OwnCloud* e *Pydio* são as principais tecnologias de *frontends* utilizadas por nuvens de armazenamento de dados. Como algumas características em comum, observa-se que elas são *open source* e dão ênfase na sincronia, armazenamento e compartilhamento de dados.

Os *frontends* estudados utilizam tecnologias de *backend* que são os componentes responsáveis pelo armazenamento de dados. Neste processo de armazenamento de dados tem-se a preocupação com a segurança da informação, por isso utiliza-se técnicas como a criptografia para auxílio nesta proteção.

A maioria das tecnologias utiliza criptografia de dados no servidor, ou seja, eles se tornam vulneráveis a ataques no processo de envio para armazenagem e uma solução para o problema é que sejam criptografados no cliente antes do envio. A exemplo da funcionalidade já existente no *StackSync,* que realiza a criptografia no cliente, o aplicativo deste trabalho foi desenvolvido para desempenhar esta mesma função mas na plataforma *OwnCloud.*

# 5 *OwnCrypt*: Um aplicativo *web* de criptografia de dados no cliente para *OwnCloud*

Como descrito, uma ameaça pertinente no armazenamento de dados em nuvem é a possibilidade de algum atacante obter acesso indevido aos dados que lá estão armazenados, uma vez que as chaves utilizadas para realizar a criptografia no servidor ficam armazenadas dentro do mesmo servidor.

De forma a mitigar esse problema, propõe-se que os dados sejam criptografados no computador do usuário antes de serem enviados para a nuvem. Dessa forma, qualquer acesso não autorizado aos arquivos armazenados na nuvem não resultará em informação útil sobre eles.

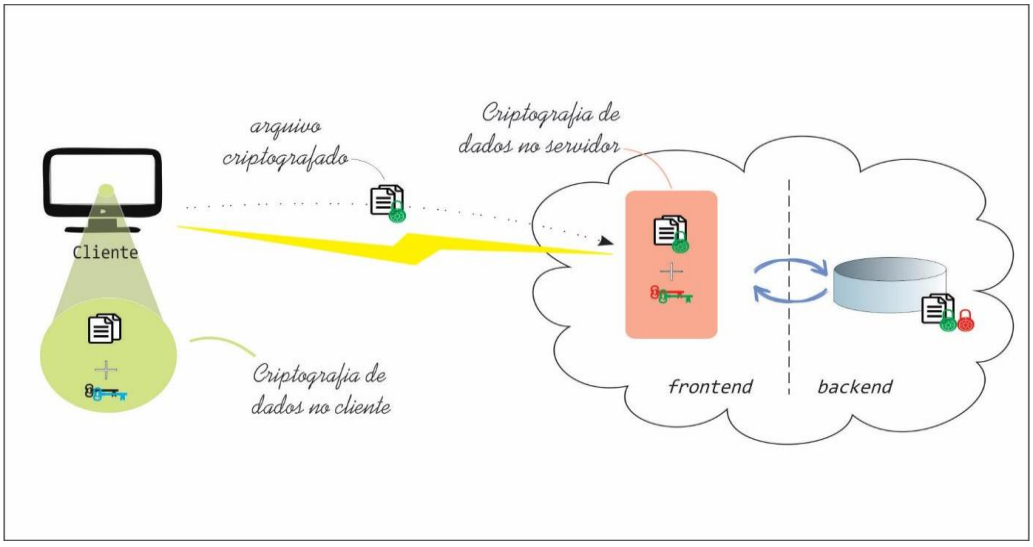
Este capítulo apresenta uma solução que possibilita a criptografia de dados no cliente. Ela possibilita que usuários criptografem seus arquivos antes de enviá-los para os servidores da nuvem.

Dentre as tecnologias para *frontend* de nuvens de armazenamento de dados apresentadas na seção 4.2, o *OwnCloud* foi escolhido para a implementação do *app*, tendo em vista que o mesmo não possui até então esta funcionalidade implementada pela comunidade e o desenvolvimento de *apps* para este *frontend* é livre.

## 5.1 Visão geral do *Owncrypt*

A solução proposta na seção anterior é exemplificada na figura 13. Onde os dados passam pelo processo de criptografia no cliente e depois são enviados através da *internet* para a nuvem de armazenamento do *OwnCloud*.

*Figura 13: Modelo de criptografia de dados no cliente.*




Fonte: Elaborada pelos autores

Pode-se notar que, caso o arquivo enviado seja acessado durante o envio ou após armazenado, ele estará protegido e o atacante não obterá o conteúdo legível.

Foi desenvolvido o *OwnCrypt* com a premissa maior de ser executado no navegador do usuário, o *app* é instalado no *frontend OwnCloud* e disponibilizado para todos os usuários da plataforma.

Em outras palavras, ao final deste trabalho, será possível realizar o armazenamento de dados em nuvens que utilizam o *OwnCloud* como *frontend* de forma mais segura.

O algoritmo de criptografia utilizado, a geração de chaves e seu armazenamento serão apresentados na seção a seguir e a lista completa dos requisitos para a implementação do app será apresentada na seção 5.3.

## 5.2 Tecnologias Empregadas

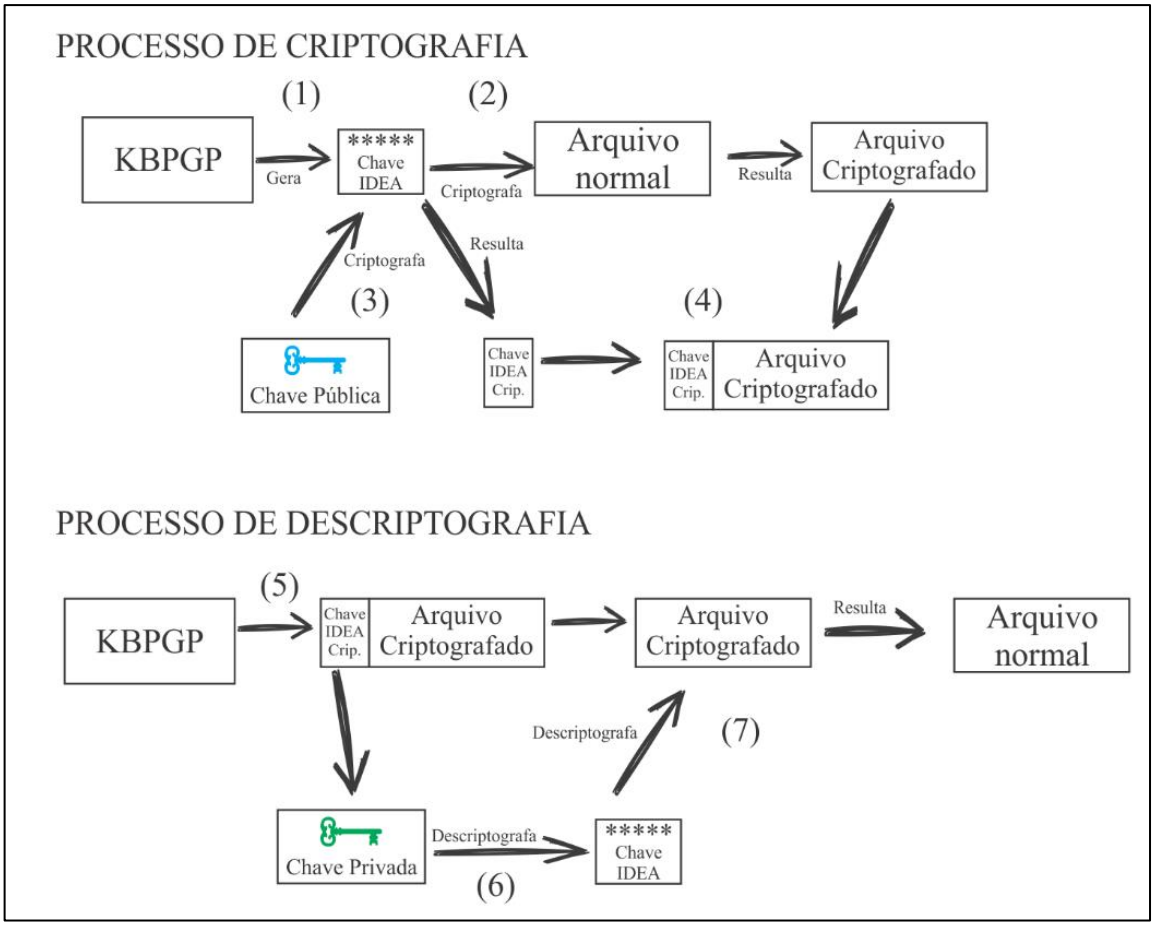
Nesta seção são apresentadas as principais tecnologias utilizadas para a implementação do *Owncrypt*.

O *app* foi desenvolvido dentro do padrão oferecido para desenvolvimento de apps para o *OwnCloud* (vide seção 4.5), ou seja, utilizando a API do *OwnCloud* para a realização do *upload* e *download* dos arquivos, no entanto, a parte de criptografia foi desenvolvida em *JavaScript*.

*JavaScript* é uma linguagem de programação interpretada diretamente pelo navegador da *internet* e é atualmente compatível com a maioria dos navegadores (FLANAGAN), ou seja, o script de criptografia é executado no computador do cliente, atendendo assim a premissa maior deste projeto.

Para enviar um arquivo para a nuvem através do *OwnCrypt*, é necessário possuir um certificado digital PGP, pois o processo de criptografia deste *app* é baseado no modelo do PGP (vide seção 2.4) conforme demonstrado na figura a seguir:

*Figura 14: Processo de criptografia e descriptografia dos dados*




Fonte: Elaborada pelos autores.

Considerado um sistema de criptografia hibrido, o PGP utiliza tanto o modelo de criptografia simétrica quanto o modelo de criptografia assimétrica. Ele gera (1) aleatoriamente uma chave simétrica, denominada chave IDEA, a qual fará a criptografia do arquivo enviado pelo usuário (2). Em seguida a chave IDEA é criptografada (3) com a chave pública do usuário e então anexada ao arquivo criptografado (4), que por fim será salvo.

Para realizar o processo contrário, o PGP lê (5) o arquivo salvo, separa a chave IDEA criptografada do arquivo criptografado e com a chave privada do usuário, descritptografa-a (6). Em seguida utiliza esta chave para descriptografar o arquivo (7) e entregá-lo ao usuário.

O algoritmo utilizado para realizar a criptografia dos dados é o AES cuja chave simétrica de criptografia é definida aleatoriamente e distribuída conforme descrito no parágrafo anterior.

Para desenvolvimento do *script* de criptografia utilizado no *OwnCrypt*, foi utilizada a biblioteca denominada *KBPgp* [Disponível em https://keybase.io ] a qual implementa o protocolo *OpenPGP* definida na RFC 4880.

## 5.3 Arquitetura do *OwnCrypt*

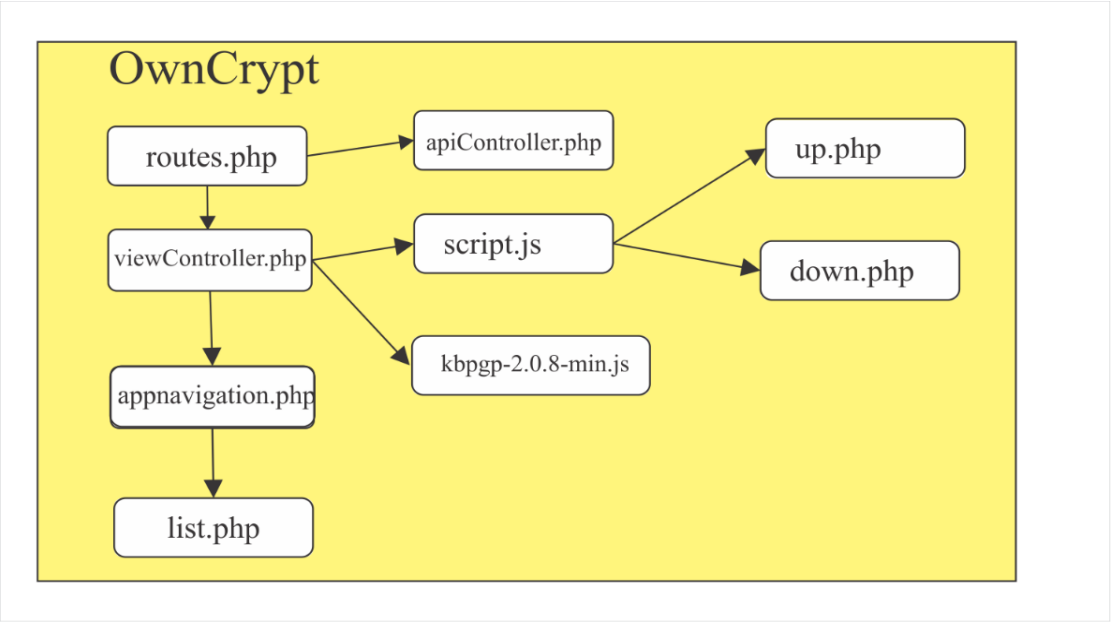
Nesta seção é apresentada a arquitetura do *OwnCrypt*, quais arquivos são utilizados e seus papéis na lógica do *app*.

Conforme citado na seção anterior, a implementação da parte de criptografia foi realizada em *JavaScript*, no entanto, as funcionalidades de listagem dos arquivos, *upload* e *download* são implementadas em PHP.

A fim de manter o padrão de armazenamento e tratamento dos arquivos que são armazenados no *OwnCloud*, foi utilizado o app padrão “files”. Este app é nativo do *OwnCloud* e realiza todas as funções relativas aos arquivos, como salvar, baixar, mover, apagar etc.

A figura 15 apresenta o esquema geral de relações entre os arquivos do *app*.

*Figura 15: Arquitetura geral do OwnCrypt*




Fonte: Elaborada pelos autores

## 5.4 Implementação

Nesta seção serão apresentados detalhes sobre a implementação do *OwnCrypt*, seus requisitos, os métodos que cada arquivo implementa e suas funções dentro do *app*.

### 5.4.1 Requisitos

Requisitos de software nada mais são do que um conjunto de atividades que o software deve desempenhar, com suas limitações e restrições, além de características não ligadas diretamente às funções desempenhadas pelo software (SOMMERVILLE, 2011).

- **Requisitos funcionais**

Os requisitos funcionais são responsáveis por descrever claramente as funções do sistema, ou seja, o que o sistema deve ou não fazer. Tais especificações devem ser completas e consistentes sem a possibilidade de contradição de informação. O Quadro 2 apresenta os requisitos funcionais no *app* desenvolvido.

*Quadro 2 - Requisitos funcionais do OwnCrypt*

| Requisito | Ator | Descrição |
|---|---|---|
| RF 01 - Ser compatível com o *OwnCloud* | Desenvolvedor | O *app* deve ser escrito de modo compatível com a plataforma *OwnCloud*, de forma que possa ser distribuído como uma extensão da mesma. |
| RF 02 - Realizar a criptografia dos dados no cliente. | Desenvolvedor | Os dados devem ser criptografados antes de serem enviados para serem armazenados na nuvem. |
| RF 03 - Permitir que o usuário faça *upload* dos arquivos. | Usuário | O usuário deve ser capaz de enviar seus arquivos para serem armazenados na nuvem. |
| RF 04 - Permitir que o usuário faça *download* dos arquivos. | Usuário | O usuário deve ser capaz de fazer o *download* dos arquivos. |
| RF 05 - Ser executado em um navegador *web.* | Desenvolvedor | Seu acesso é feito através de acesso com uma página *Web.* |
| RF 06- Permitir que o usuário visualize os arquivos. | Usuário | Quando o app é acessado através do navegador, o usuário poderá visualizar a lista dos seus arquivos que estão armazenados na nuvem. |

Fonte: Elaborada pelos autores

Os requisitos não funcionais são restrições sob as quais os sistemas devem operar, tais restrições podem ser observadas durante o desenvolvimento do aplicativo, como por exemplo, a linguagem de programação que foi utilizada para o desenvolvimento, versões compatíveis de navegadores, plataformas e sistemas operacionais compatíveis, etc. No Quadro 2 visualiza-se estes requisitos presentes no *app*.

*Quadro 3 Requisitos não-funcionais do OwnCrypt*

| | |
|---|---|
| **RNF 01** | O *app* deverá ser acessado via *web* browser, devendo ser compatível com os navegadores: EDGE, Firefox (a partir do 45), Safari (a partir do 10) e Chrome. |
| **RNF 02** | O *app* só poderá enviar arquivos após seres criptografados para a nuvem. |
| **RNF 03** | O sistema deve ser implementado utilizando a linguagem *JavaScript*. |

Fonte: Elaborada pelos autores

### 5.4.2. Arquivos e funções do *OwnCrypt*

Conforme apresentado na seção 4.5.2, o *OwnCloud* possui uma estrutura padrão para a criação de *apps*. A seguir serão apresentados os principais arquivos e as funções utilizados para a implementação do *OwnCrypt*.

- **Arquivo /owncrypt/appinfo/routes.php**

O arquivo *routes.php* implementa as rotas do *app*, tais rotas são conectadas através dos métodos definidos nos *controllers*.

Tais rotas devem ser passadas em um array para o método registerRoutes, conforme imagem a seguir:

*Figura 16: Arquivo routes.php*

```
22  $application = new Application();
23  $application->registerRoutes($this,[
24          'routes' => [
25              ['name' => 'view#index', 'url' => '/', 'verb' => 'GET'],
26
27              ]
28  ]);
```

Fonte: Elaborada pelos autores

O parâmetro ‘*name’* define que o método controlador *‘index’* está no *controller viewController*. O parâmetro *‘url’* define a url que aparecerá após */index.php/apps/owncrypt.* Por fim, o parâmetro ‘*verb’* define o método HTTP que deve ser utilizado para solicitar a página.

- **Arquivo /owncrypt/lib/viewController.php**

O arquivo *viewController.php* faz parte da classe dos *controllers*, os quais são responsáveis pela lógica central do *app*. No caso do *OwnCrypt*, ele é responsável pela configuração de exibição do *app*.

O método index define a exibição da página inicial do *app*, e importa os scripts utilizados na mesma. É neste arquivo que o arquivo *owncrypt.js*, cujo funcionamento será detalhado posteriormente, é importado, conforme demonstrado na figura 16.

*Figura 17: Importação dos scripts do OwnCrypt*

```
193         \OCP\Util::addscript('owncrypt', 'jquery.easyModal');
194         \OCP\Util::addscript('owncrypt', 'kbpgp-2.0.8-min');
195         \OCP\Util::addscript('owncrypt', 'owncrypt');
```

Fonte: Elaborada pelos autores

Também são importados os arquivos *jquery.easyModal.js* e *kbpgp-2.0.8-min.js*, que são as bibliotecas para exibir os *modals* onde são exibidos os formulários para *upload* e *download* dos arquivos e a biblioteca criptográfica, respectivamente.

O comando \OCP\Util::addscript(‘app’, ‘script’) é próprio da API do OwnCloud, conforme apresentado na seção 2.5.

- **Arquivo /owncrypt/lib/apiController.php**

O arquivo *apiController.php* também faz parte da classe dos *controllers*, de maneira análoga ao *viewController.php* apresentado acima. Nele estão configurados os métodos de gerenciamento dos arquivos, tais como exibição em miniaturas, ordenação da lista de arquivos etc.

Nenhuma alteração foi realizada neste arquivo para realização deste trabalho.

- **Arquivo /owncrypt/template/appnavigation.php**

É neste arquivo que está configurado o código HTML exibido na página do *app*. As imagens a seguir demonstram os dois <*div*> contendo os <*input*> que são utilizados para receber os arquivos e as chaves para *upload* e *download*, respectivamente.

*Figura 18:  owncrypt-upload*

```
<div id = "owncrypt-upload">
    <h1>Iniciar Upload Com OwnCrypt</h1>
    <button id="myCryptButton">Iniciar Upload</button>
        <div id="cryptModal">
            <h1>Upload seguro</h1>
            <div id = "FileLoader">
                <form action="#">
                    <p>Agora escolha seu arquivo para upload seguro</p>
                    <input type="file" name="userFile" id="userFile" />
                    <button type="submit" id="submitFile">OK</button>
                </form>
            </div>
        </div>  <!-- fim do cryptModal -->
        <div id = "PKeyLoader">
            <p>Carregar chave pública para criptografia</p>
            <input type = "file" name="userPublicKey" id="userPublicKey">
            <button type="submit" id="submitPublicKey">Carregar</button>
        </div>
</div>      <!-- fim do owncrypt-upload -->
```

Fonte: Elaborada pelos autores

O * owncrypt-upload* contém dois **modals**, o primeiro *"cryptModal"* é responsável por receber o arquivo que será criptografado e enviado para a nuvem. O segundo *"PKeyLoader"* recebe o arquivo da chave pública do certificado digital PGP do usuário.

*Figura 19: owncrypt-download*

```
<div id = "owncrypt-download">
    <div id = "decryptModal">
        <h1>Download</h1>
        <div id = "SKeyLoader" >
            <p>Carregar chave privada para descriptografia de arquivo</p>
            <input type = "file" name="userSecretKey" id="userSecretKey">
            <button type="submit" id="submitSecretKey">Carregar</button>
        </div>
    </div><!-- fim do decryptModal -->
        <div id = "pswLoader">
            <p>Insira a senha de seu certificado</p>
            <form name="form1" action="#">
                <input type="password" name="psw">
                <button type="submit" id="submitPsw">Ok</button>
            </form>
        </div>
        <input type="hidden" name="enc_field">
</div><!-- fim do owncrypt-download -->
```

Fonte: Elaborada pelos autores

O * owncrypt-download* contém os *modals* para informação do arquivo de chave privada do certificado digital PGP do usuário e a senha para ler tal arquivo. Nele também está contido um *<input>* oculto *"enc_field"*, o qual receberá o conteúdo do arquivo criptografado para realizar o download.

Todos os elementos citados anteriormente são manipulados pelo script owncrypt.js para realizar a criptografia, descriptografia, *upload* e *download*. O detalhamento de tais funções será descrito adiante nesta seção.

- **Arquivo /owncrypt/template/list.php**

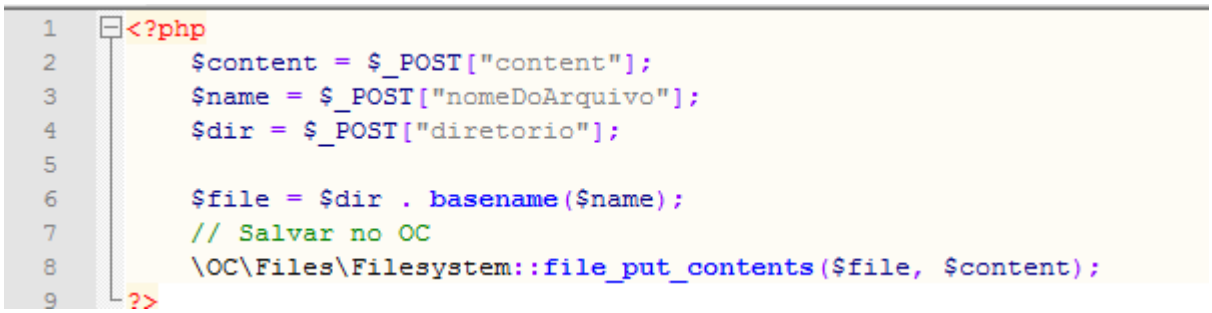
Este arquivo é responsável por listar os arquivos salvos na nuvem em forma de lista. Nenhuma alteração foi realizada neste arquivo para desenvolvimento deste trabalho.

- **Arquivo /owncrypt/ajax/up.php**

Este arquivo é responsável por receber o arquivo criptografado e em seguida salva-lo na nuvem.

Através de uma requisição AJAX, contida no script owncrypt.js, este arquivo recebe na variável $_POST o conteúdo e o nome do arquivo que será salvo. Utilizou-se um método da API do *OwnCloud* para realizar o salvamento.

*Figura 20: Código do arquivo up.php*



```php
<?php
    $content = $_POST["content"];
    $name = $_POST["nomeDoArquivo"];
    $dir = $_POST["diretorio"];

    $file = $dir . basename($name);
    // Salvar no OC
    \OC\Files\Filesystem::file_put_contents($file, $content);
?>
```

Fonte: Elaborada pelos autores

O método **\OC\Files\Filesystem::file_put_contents($file, $content)** cria o arquivo na nuvem com o nome e o conteúdo passado como parâmetro. O nome do arquivo deve conter o caminho relativo onde o arquivo será salvo: */teste.zip*, por exemplo.

- **Arquivo /owncrypt/ajax/down.php**

Responsável por ler o arquivo criptografado que está salvo na nuvem e enviá-lo em uma resposta para o browser com o arquivo lido, como um elemento do DOM.

O método **\OC\Files\Filesystem::readfile($filename)** lê o arquivo salvo no servidor e imprime o seu conteúdo. No trecho demonstrado na figura 20, é observado que o resultado é impresso na * enc-data*, este resultado é recuperado através de uma requisição POST e inserido no *<input> enc_field*, conforme descrito anteriormente.

*Figura 21: Código do arquivo down.php*

```
<?php $filename = $_POST["nome"]; ?>
<section>
    <div id="enc-data">
        <?php \OC\Files\Filesystem::readfile( $filename ); ?>
    </div>
</section>
?>
```

Fonte: Elaborada pelos autores

- **Arquivo kbpgp-2.0.8-min.js**

Biblioteca criptográfica que será utilizada em */owncrypt/js/owncrypt.js*.

- **Arquivo /owncrypt/js/owncrypt.js**

Este é o arquivo principal do app, ele possui três partes: leitura, criptografia e transferência. Na primeira parte do script, estão contidas as funções que gerenciam os *modals* e os *handlers* dos eventos de leitura dos arquivos. Para ler os arquivos é utilizada a API *FileReader*.

Segundo MDN:

> O objeto *FileReader* permite aplicações *web* ler assincronamente o conteúdo dos arquivos (ou *buffers* de dados puros) do computador do usuário, utilizando o objeto *File* ou *Blob* para especificar o arquivo ou os dados a serem lidos. Objetos *File* podem ser obtidos a partir de um objeto *FileList* retornado como resultado da seleção de um usuário utilizando um elemento <*input*>, a partir de uma operação de *drag and drop DataTransfer*, ou a partir de um *mozGetAsFile()* da api *HTMLCanvasElement*.

A seguir são mostrados os trechos de código que são responsáveis pela parte de leitura do script, explicada acima.

*Figura 22: Configuração dos modals*

```
16          $('#cryptModal').easyModal({
17              overlay: 0.4,
18              overlayClose: true,
19          });
20      });
21      $(function() {
22          $('#decryptModal').easyModal({
23              overlay: 0.4,
24              overlayClose: true,
25          });
26      });
27      $(function() {
28          $('#PKeyLoader').easyModal({
29              overlay: 0.4,
30              overlayClose: true,
31          });
32      });
33      $(function() {
34          $('#pswLoader').easyModal({
35              overlay: 0.4,
36              overlayClose: true,
37          });
38      })
```

Fonte: Elaborada pelos autores

*Figura 23: Handler de evento click do botão "Iniciar Upload"*

```
39      $('#myCryptButton').click(function(){
40          $('#cryptModal').trigger('openModal');
41          return false;
42      });
```

Fonte: Elaborada pelos autores

*Figura 24: Handler de evento click do botão SubmitFile.*

```
50      //Leitura do arquivo a ser criptografado
51      $("#submitFile").click(function(){
52          var x = document.getElementById("userFile");
53          formData.append("nomeDoArquivo", x.value.split('\\').pop().split('/').pop());
54          var reader = new FileReader();
55          reader.onloadend = function(e){
56              if (e.target.readyState == FileReader.DONE) {
57                  saida = e.target.result;
58                  if(saida != ""){
59                      $('#PKeyLoader').trigger('openModal');
60                  }
61              }
62          };
63          reader.readAsText(x.files[0]);
64          $('#cryptModal').trigger('closeModal');
65          $('#PKeyLoader').trigger('openModal');
66          return false;
67      });
```

Fonte: Elaborada pelos autores

Pode-se observar na linha 54 da imagem acima, onde é instanciado o objeto *FileReader* que fará a leitura do arquivo enviado e guardará o seu conteúdo na variável “saida”.

*Figura 25: Handler de evento do botão submitPublicKey.*

```
68      //Leitura da Chave Publica
69      $("#submitPublicKey").click(function(){
70          var pKey = "";
71          var reader = new FileReader();
72          var k = document.getElementById("userPublicKey");
73          reader.onloadend = function(e){
74              if (e.target.readyState == FileReader.DONE) {
75                  pKey = e.target.result;
76                  if (pKey != "" ){
77                      encryptFile(saida, pKey);
78                      $('#PKeyLoader').trigger('closeModal');
79                  }
80              }
81          };
82      reader.readAsText(k.files[0]);
83      });
```

Fonte: Elaborada pelos autores

No trecho acima, caso a leitura da chave pública seja realizada com sucesso, ela é passada como parâmetro juntamente com a saída da função anterior para a função *encryptFile*.

A seguir são apresentados os trechos do código os quais são responsáveis por ler a chave privada e a senha da mesma para o processo de descriptografia do arquivo.

*Figura 26: Handler do evento do botão submitSecretKey*

```
85      $("#submitSecretKey").click(function(){
86          var reader = new FileReader();
87          var k = document.getElementById("userSecretKey");
88          reader.onloadend = function(e){
89              if (e.target.readyState == FileReader.DONE) {
90                  saidaK = e.target.result;
91                  if (saidaK != "" ){
92                      $('#decryptModal').trigger('closeModal');
93                      $('#pswLoader').trigger('openModal');
94                  }
95              }
96          };
97      reader.readAsText(k.files[0]);
98      });
```

Fonte: Elaborada pelos autores

*Figura 27: Handler do evento do botão submitPsw*

```
100     $("#submitPsw").click(function(){
101         var pws = form1.psw.value;
102         var encrypted = "";
103         console.log('pws = '+pws+'   encrypted= '+encrypted);
104         decryptFile(encrypted, saidaK, pws);
105         return false;
106     });
```

Fonte: Elaborada pelos autores

O arquivo criptografado é recuperado do servidor através de uma requisição POST e armazenado na variável *"encrypted"*, a qual é enviada como parâmetro, juntamente com a senha e a chave privada, para a função *decryptFile*.

A segunda parte do script é responsável por realizar a criptografia propriamente dita. As funções encryptFile e decryptFile utilizam a biblioteca KBPGP apresentada na seção 5.2 para realizar a criptografia e a descriptografia, conforme demostrado nas próximas figuras.

*Figura 28: Função encryptFile*

```
108     function encryptFile(file, pKey){
109         kbpgp.KeyManager.import_from_armored_pgp({
110             armored: pKey
111         }, function(err, km) {
112             if (!err) {
113             }
114             var params = {
115                 msg: file,
116                 encrypt_for: km
117             };
118             kbpgp.burn(params, function(err, result_string, result_buffer) {
119                     formData.append("content",result_string);
120                     saveFile(formData);
121             });
122         });
123     };
```

Fonte: Elaborada pelos autores

Na imagem acima pode-se observar na linha 109 a utilização da função *kbpgp.KeyManager.import_from_armored_pgp*, esta função lê a chave PGP e cria um *"key manager"* o qual será utilizado para realizar a criptografia. De acordo com o explicado na seção 5.2.

O método que realiza a criptografia é o *"kbpgp.burn"*. Ele recebe o objeto *"params"* como parâmetro, onde consta a mensagem a ser criptografada, pode ser uma *String* ou um *buffer*, e o *key manager*. Ele retorna duas saídas, uma S*tring* e um *buffer*. Neste trabalho é utilizada a saída como *String* para ser enviada ao servidor.

*Figura 29: Função decryptFile*

```
125     function decryptFile(file, sKey, psw){
126         kbpgp.KeyManager.import_from_armored_pgp({armored: sKey
127         }, function(err, km) {
128           if (!err) {
129             if (km.is_pgp_locked()) {
130               km.unlock_pgp({
131                 passphrase: psw
132               }, function(err) {
133                 if (!err) {
134                   console.log("Chave privada carregada com senha");
135                 }
136               });
137             } else {
138               console.log("Chave privada carregada sem senha");
139             }
140           }
141           //decrypt here
142           var ring = new kbpgp.keyring.KeyRing;
143           var kms = [km];
144           var pgp_msg = file;
145           for (var i in kms) {
146             ring.add_key_manager(kms[i]);
147           }
148           kbpgp.unbox({keyfetch: ring, armored: pgp_msg }, function(err, literals) {
149             if (err != null) {
150               return console.log("Problem: " + err);
151             } else {
152               downloadFile(literals[0], name, type);
153               }
154           return false;
155           });
156         });
157     };
```

Fonte: Elaborada pelos autores

A função *decryptFile* recebe o arquivo criptografado, a chave privada e a senha do certificado como parâmetro. De forma semelhante ao *encryptFile*, um *key manager* é criado para então realizar a descriptografia. A função *kbpgp.unbox* é utilizada para descriptografar e tem como saída um objeto *Literals*. Este objeto pode ser lido como *String* ou como *buffer*.

Neste trabalho, utilizamos a saída como buffer para construir um arquivo e realizar o *download* para o usuário.

Na terceira parte do script temos as funções *saveFile*, *readFile* e *downloadFile*, responsáveis por enviar o arquivo criptografado ao servidor, solicitar um arquivo criptografado ao servidor e salvar o arquivo descriptografado no computador do usuário, respectivamente.

*Figura 30: Função saveFile*

```
159     function saveFile(fd){
160         var request = new XMLHttpRequest();
161         request.open('post','/owncloud/index.php/apps/owncrypt/ajax/up.php');
162         request.send(fd);
163     };
```

Fonte: Elaborada pelos autores

A função *saveFile* utiliza um objeto *XMLHttpRequest* para fazer a requisição POST para o servidor.

Como pode ser verificado no MDN, *XMLHttpRequest* é uma API que fornece funcionalidade ao cliente para transferir dados entre um cliente e um servidor. Ele fornece uma maneira fácil de recuperar dados de um URL sem ter que fazer uma atualização de página inteira. Isso permite que uma página da *Web* atualize apenas uma parte do conteúdo sem interromper o que o usuário esteja fazendo. *XMLHttpRequest* é usado constantemente na programação de AJAX.

*XMLHttpRequest* foi originalmente projetado pela Microsoft e adotado pela Mozilla, Apple e Google. Está sendo padronizado pela WHATWG (*Web Hypertext Application Technology Working Group*). Apesar do nome, *XMLHttpRequest* pode ser usado para recuperar qualquer tipo de dado, e não apenas XML, suportando também protocolos diferentes de HTTP (incluindo *file* e ftp ).

*Figura 31: Funcão readFile.*

```
165     function readfile(name) {
166         $.ajax({
167             type: 'POST',
168             dataType: 'html',
169             url: '/owncloud/index.php/apps/owncrypt/ajax/down.php',
170             beforeSend: function(){
171                 // $("#dados").html("Carregando...");
172             },
173             data: {nome:name},
174             success: function (msg){
175                 $("#enc_field").html(msg); //passa retorno para objeto no html
176                 $('#decryptModal').trigger('openModal');
177             }
178         });
179     };
180
181 });
```

Fonte: Elaborada pelos autores

A função *readFile* realiza uma requisição POST para o servidor e obtém um *html* como resposta, este *html* possui o arquivo lido como *string*, em seguida guarda este valor em uma variável e chama a função *downloadFile*, a qual salva o arquivo no computador do usuário

*Imagem 31. Função downloadFile*

```
183  function downloadFile(data, filename, type) {
184      var a = document.createElement("a"),
185          file = new Blob([data], {type: type});
186      if (window.navigator.msSaveOrOpenBlob) // IE10+
187          window.navigator.msSaveOrOpenBlob(file, filename);
188      else {
189          var url = URL.createObjectURL(file);
190          a.href = url;
191          a.download = filename;
192          document.body.appendChild(a);
193          a.click();
194          setTimeout(function() {
195              document.body.removeChild(a);
196              window.URL.revokeObjectURL(url);
197          }, 0);
198      }
199  }
```

]

Fonte: Elaborada pelos autores

A figura acima demonstra como o arquivo é preparado para *download*. Recebendo as informações do arquivo que foram passadas pelo método *decryptFile*, o *buffer*, o nome do arquivo e o tipo.

Desta forma é possível garantir que todo arquivo enviado para a nuvem é cifrado ainda no navegador do usuário, da mesma forma garante-se que o arquivo seja decifrado no navegador, ocultando do servidor quaisquer informações sobre o conteúdo do arquivo original. O servidor da nuvem apenas armazena um conjunto de dados não legíveis.

Na próxima seção é apresentado um exemplo do uso do *app OwnCrypt*.

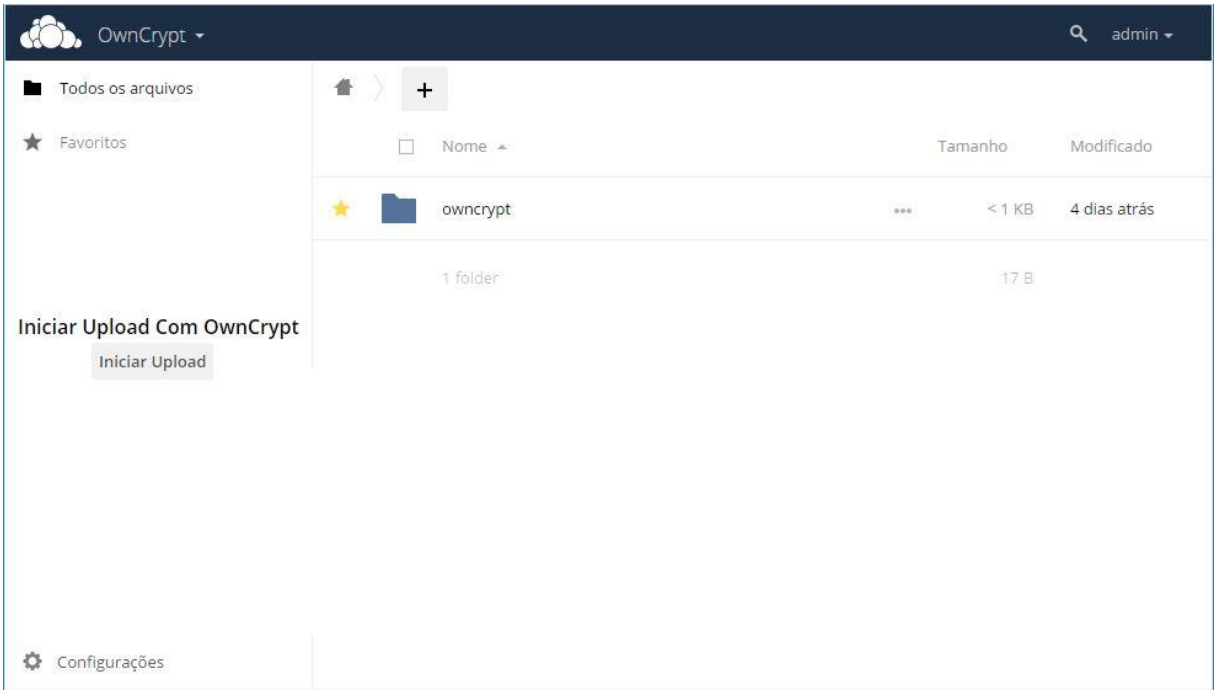
## 5.5 Exemplo de uso do *OwnCrypt*

Esta seção demonstra como é realizada a criptografia de um arquivo “teste.txt” com as imagens das telas que o usuário interage.

Conforme apresentado no quadro de requisitos, o requisito RF01 - Ser Compatível com o *OwnCloud* - o *app* é acessado dentro do próprio *OwnCloud*, através do acesso *Web*.

No canto superior esquerdo, possui um menu onde é possível alternar entre os apps instalados.

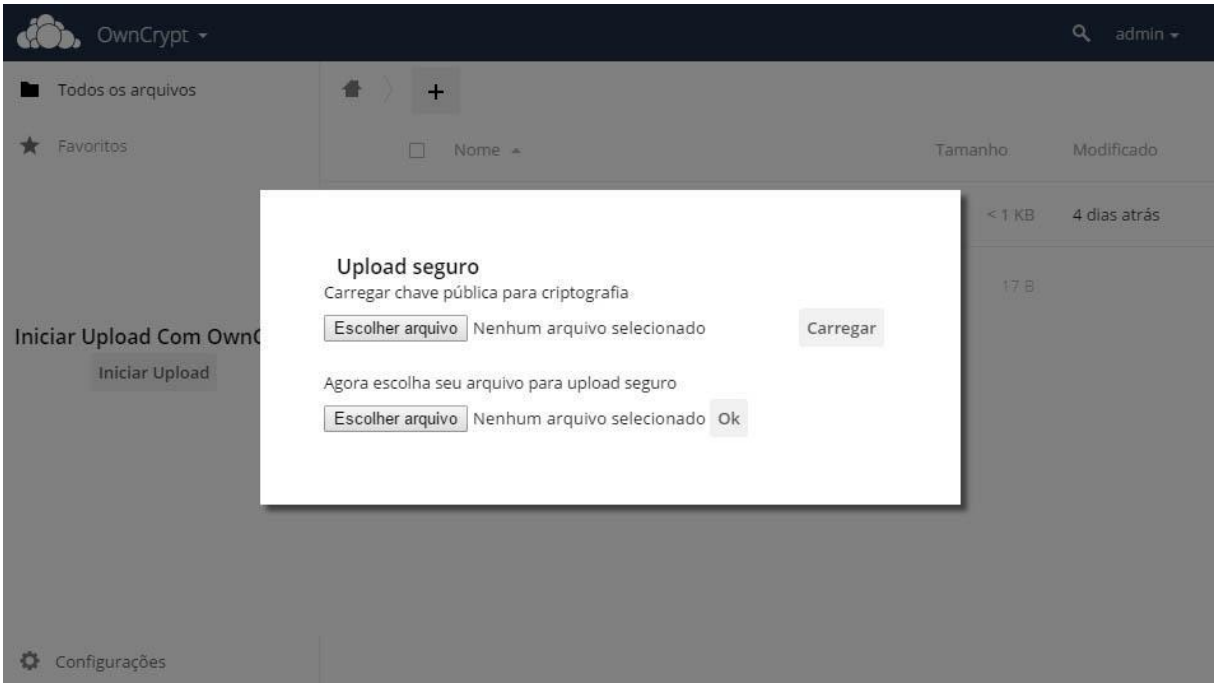
*Figura 32: Tela inicial do app OwnCrypt.*




Fonte: Elaborada pelos autores

Para realizar o *upload* através do *app OwnCrypt*, deve-se acionar o botão Iniciar *Upload*, localizado na barra lateral. Este botão abrirá um modal que é responsável por receber as entradas necessárias: o arquivo contendo a chave pública do usuário, em formato ASCII, e o arquivo que será criptografado é enviado posteriormente. A figura 16 apresenta esta tela de carregamento.

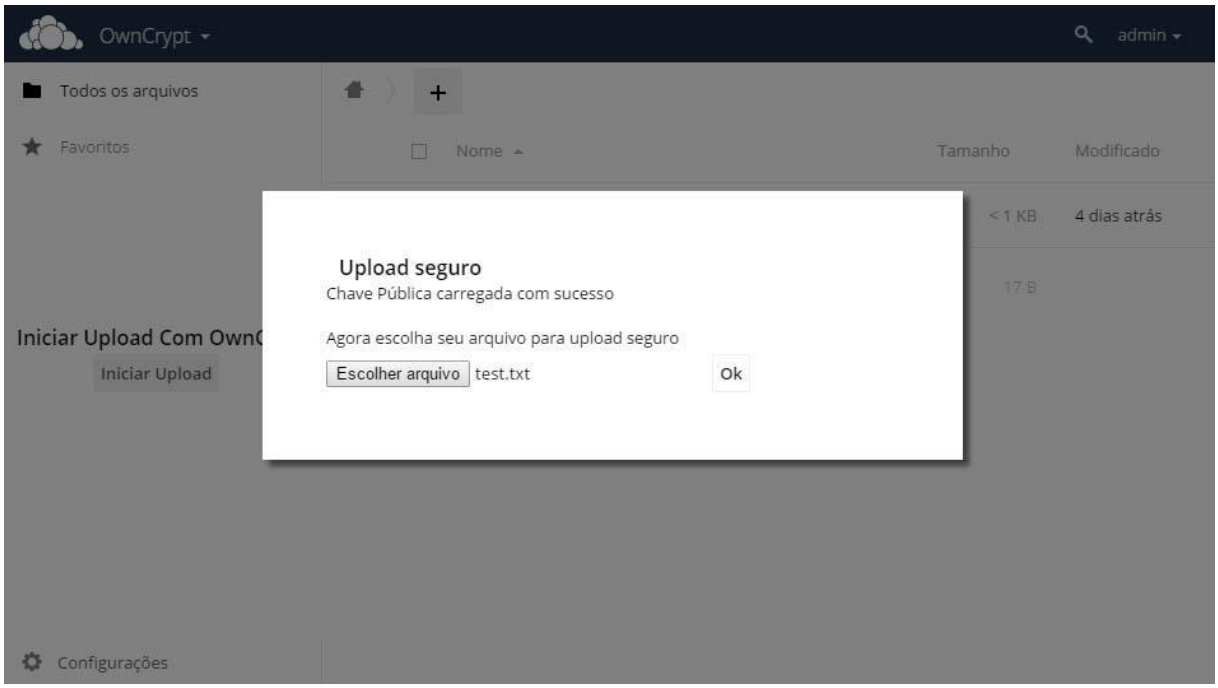
*Figura 33: OwnCrypt: Carregando a chave pública.*




Fonte: Elaborada pelos autores

É importante ressaltar que é necessário carregar a chave pública antes de enviar o arquivo para *upload*. Para garantir que a chave pública esteja carregada, o *app* faz uma validação antes de executar o processo de *upload*.

*Figura 34: OwnCrypt: Envio de arquivo.*

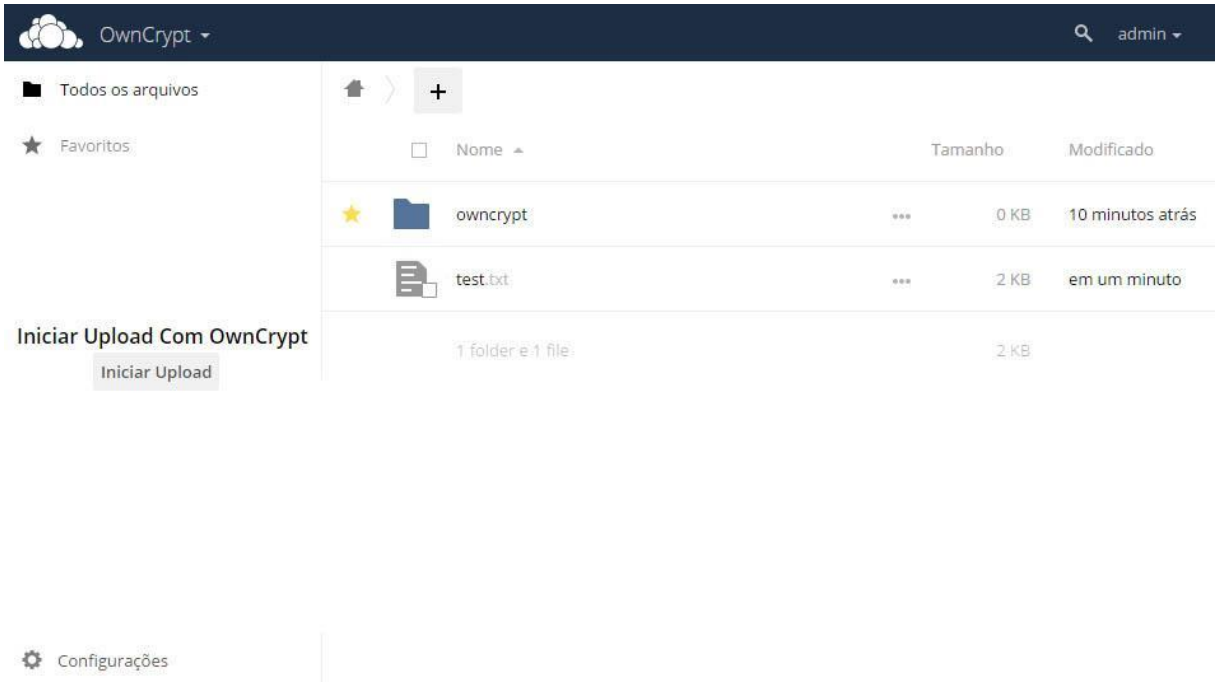



Fonte: Elaborada pelos autores

Após a conclusão da criptografia, o arquivo é enviado para o armazenamento e automaticamente é listado na pasta correspondente, exibida no lado direito da janela.

A figura 35 apresenta um exemplo de tal processo.

*Figura 35: OwnCrypt: Upload concluído.*

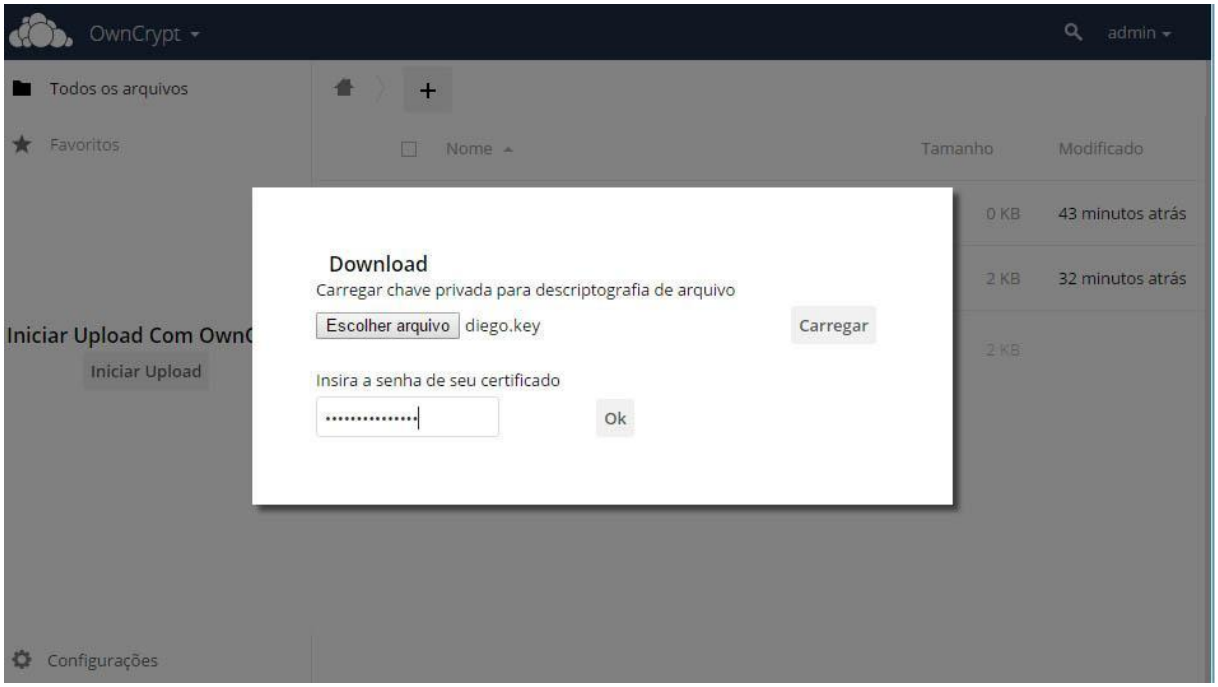



Fonte: Elaborada pelos autores

Para realizar o processo de *download*, o usuário clica no arquivo desejado e escolhe a opção *Download*. Um modal é apresentado solicitando a inserção da chave privada e da senha do certificado para realizar a descriptografia do arquivo e então enviá-lo para *download*.

Na figura 36 é apresentada a tela onde é realizada essa operação.

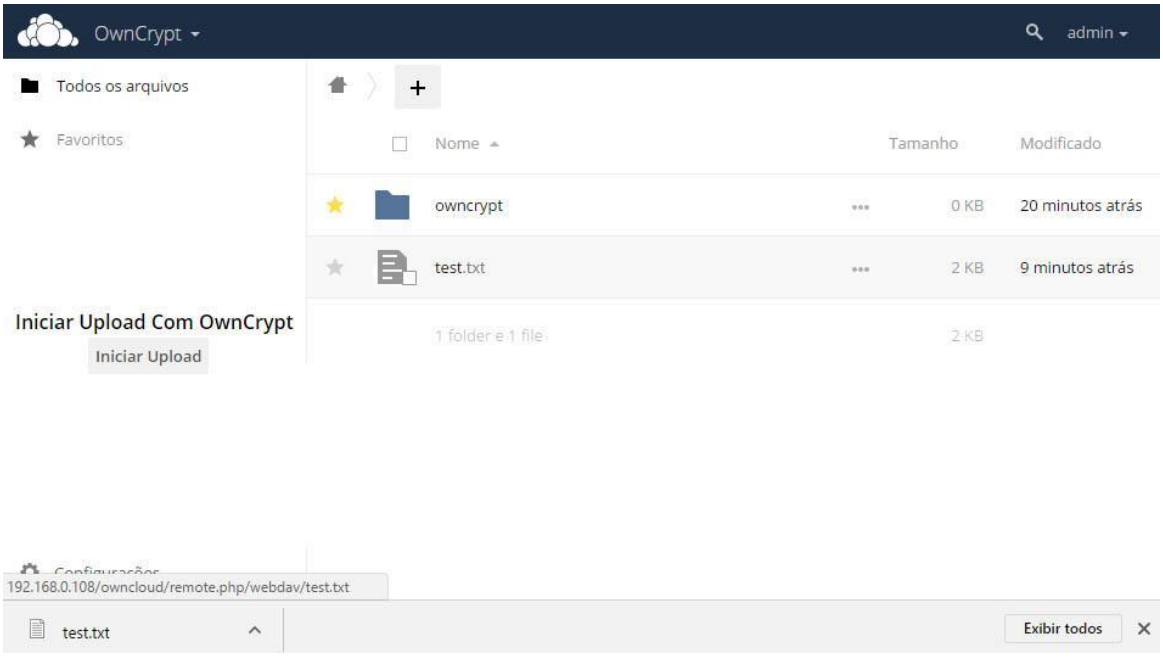
*Figura 36: OwnCrypt: Formulário de leitura da Chave Privada e senha.*




Fonte: Elaborada pelos autores

Após a descriptografia o *download* é realizado normalmente, conforme a configuração do navegador do usuário. A figura a seguir, demonstra o *download.*

*Figura 37: OwnCrypt: Download concluído*

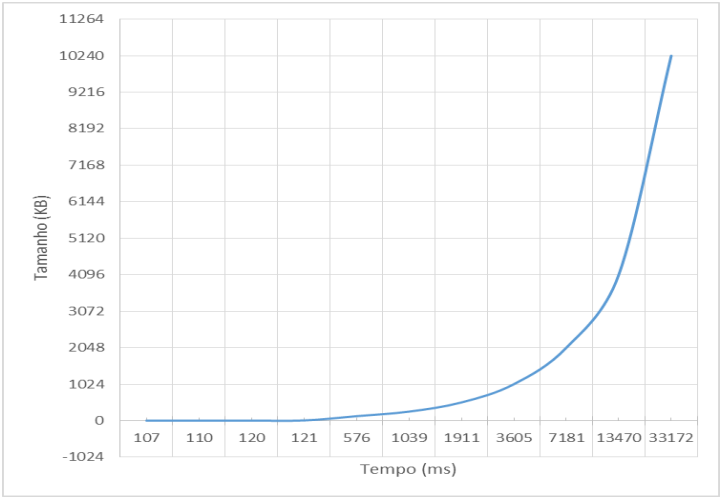



Fonte: Elaborada pelos autores

## 5.6 Testes Realizados

A figura a seguir foi obtida através de testes de desempenho realizados para medir o tempo necessário para criptografar arquivos de diferentes tamanhos. Para fins de teste, foram utilizados arquivos de tamanho 1KB, 2KB, 4KB, 8KB, 128KB, 256KB, 512KB, 1MB, 2MB, 4MB e 10MB, obtendo o tempo, respectivamente, de 107ms, 110ms, 120ms, 121ms, 576ms, 1039ms, 1911ms, 3605ms, 7181ms, 13470ms e 33172ms.

*Figura 38 Gráfico Tamanho x Tempo de criptografia*




Fonte: Elaborada pelos autores

## 5.7 Considerações Finais

No capítulo apresentado verifica-se como o *OwnCrypt* pode trazer mais segurança para o armazenamento de dados em nuvens baseadas no *OwnCloud*, pois desta forma é possível garantir que todo arquivo enviado para a nuvem é cifrado ainda no navegador do usuário, da mesma forma garante-se que o arquivo seja decifrado no navegador, ocultando do servidor quaisquer informações sobre o conteúdo do arquivo original. O servidor da nuvem apenas armazena um conjunto de dados não legíveis.

# 6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho gerou um *app* para a plataforma do *OwnCloud* que permite criptografar arquivos antes de enviá-los para a nuvem de armazenamento, funcionalidade esta que não estava presente nesta plataforma até então, trazendo assim real contribuição para a segurança deste *frontend*.

A principal limitação encontrada para o desenvolvimento deste *app* é a falta de bibliotecas que implementem os algoritmos criptográficos para execução no navegador. Isto resultou no uso da biblioteca *kbpgp* apresentada no capítulo 5, no entanto, esta biblioteca ainda está em desenvolvimento, o que resulta em pouca documentação disponível.

Outra limitação encontrada foi com relação ao tamanho dos arquivos, o processo de criptografia eleva bastante o tempo utilizado para realizar o *upload* dos arquivos, de acordo como pode ser observado nos testes

Entende-se que esta é uma ferramenta que traz uma nova possibilidade para criptografia de dados, no entanto, não é aplicável para toda e qualquer situação de armazenamento de dados na nuvem. Devido ao tempo elevado que é necessário para criptografar os dados, sugere-se uma avaliação cuidadosa para empregar tal ferramenta. Sendo possível desenvolver um estudo futuro sobre o custo computacional requerido para a cifragem dos arquivos e possíveis melhorias para diminuir tais custos, talvez o particionamento dos arquivos para realizar o *upload*.

Por fim, vale ressaltar que arquivos cifrados no cliente não podem ser compartilhados, uma vez que para decifrá-los é necessária a chave privada do proprietário.

Outra sugestão de trabalho é implementar a criptografia dos dados para os demais clientes do *OwnCloud*: *Desktop Client* e *Mobile*.

## REFERÊNCIAS BIBLIOGRÁFICAS

PEDROSA, P.H.C.; NOGUEIRA, T. Computação em Nuvem**,** 2011. Disponível em < http://www.ic.unicamp.br/~ducatte/mo401/1s2011/T2/Artigos/G04-095352-120531-t2.pdf>. Acesso em: 18 de março de 2016.

Cartilha de Segurança para *Internet.* Disponível em < https://cartilha.cert.br/criptografia/>. Acesso em: 13 de abril de 2017

SALLES, M; CANIC, C; SILVA, C; KOZUMA, M; RIBEIRO, M. Comparação de aplicativos livres para android: Primeira etapa do desenvolvimento de SEGUREGENMOD. 2013. Disponível em: < https://gpopai.usp.br/wordpress/wp-content/uploads/2015/08/Resumo38_corrigido-APA.pdf >
Acesso em: 26 de abril de 2017

DIAS, A.D; SAKUDE, M.T.S. Itacripto: Uma proposta de aplicativo de criptografia para o ITA. 2008. Disponível em: < http://www.bibl.ita.br/xivencita/COMP06.pdf >
Acesso em: 26 de abril de 2017

SOUSA, F.R.C.; MOREIRA, L.O. Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios, 2010. Disponível em <http://www.academia.edu/783784/Computa%C3%A7%C3%A3o_em_Nuvem_Conceitos_Tecnologias_Aplica%C3%A7%C3%B5es_e_Desafios > Acesso em: 18 de março de 2016.

KPGM INFO. Elevating Business in the Cloud. 2014. Disponível em <http://www.kpmginfo.com/EnablingBusinessInTheCloud/downloads/7397-CloudSurvey-Rev1-5-15.pdf >. Acesso em: 05 de outubro de 2016.

SOCBLOG. Estudo mostra aumento da confiança do mercado na Computação em Nuvem. 2015. Disponível em <http://www.socblog.com.br/2015/08/estudo-mostra-aumento-da-confianca-do-mercado-na-computacao-em-nuvem/ >. Acesso em: 05 de outubro de 2016

NIST. 2011. Disponível em: < https://www.nist.gov/ >. Acesso em: 18 de março de 2016

The NIST Definition of Cloud Computing. Disponível em <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf >. Acesso em: 18 de março de 2016.

JANSEN, W.; GRANCE, T**. Guidelines on security and privacy in public cloud computing,** 2011. Disponível em **<**http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-144.pdf **>.** Acesso em: 20 de janeiro de 2016.

OLIVEIRA, E.S.C.; BONETTI, T.P.; GERMANO, R. Uma visão geral da computação em nuvem, 2014. Disponível em <http://web.unipar.br/~seinpar/2014/artigos/graduacao/Ederson%20dos%20Santos%20II.pdf >. Acesso em: 18 de março de 2016.

SOUSA, F.R.C.; MOREIRA, L.O.; MACHADO, J.C. Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios. 2009. Disponível em: <https://www.researchgate.net/profile/Javam_Machado/publication/237644729_Computacao_em_Nuvem_Conceitos_Tecnologias_Aplicacoes_e_Desafios/links/56044f4308aea25fce3121f3.pdf >. Acesso em: 06 de outubro de 2016.

DIÓGENES, Y. Cap. 17- Considerações quanto à segurança na computação na nuvem, 2012. Disponível em < https://chapters.cloudsecurityalliance.org/brazil/files/2012/04/cap17.pdf >. Acesso em: 18 de março de 2016.

BORGES, H.P.; SOUZA, J.N.; SCHULZE, B.; MURY, A.R. Computação em Nuvem, 2010. Disponível em: <http://livroaberto.ibict.br/bitstream/1/861/1/COMPUTA%C3%87%C3%83O%20EM%20NUVEM.pdf >. Acesso em: 30 de Janeiro de 2017.

CASTRO, R.C.C. de; PIMENTEL, V.L. Cloud Computing: Governança e Gerenciamento de Riscos de Segurança, 2011. Disponível em: <http://www.infobrasil.inf.br/userfiles/26-05-S5-1-68740-Seguranca%20em%20Cloud.pdf>. Acesso em: 20 de agosto de 2016.

BENATALLAH, B.; WANG, L.; RANJAN, R.; CHEN, J. Cloud Computing: methodology, systems and applications**.** CRC press, 2012, p.392-399.

Computação em Nuvem: Papéis e Cenários na Nuvem. Disponível em: < http://web.ist.utl.pt/ist167662/2_1_1_2.html >. Acesso em: 20 de agosto de 2016

VECCHIOLA, C. PANDEY, S. BUYYA, R. High Performance Cloud Computing: A view of Scientific Applications. 2009.< https://arxiv.org/ftp/arxiv/papers/0910/0910.1979.pdf > . Acesso em: 27 de abril de 2017

HAYES, B. Communications of the ACM, Vol. 51 No. 7, Pages 9-11, 2008. Disponível em: <http://delivery.acm.org/10.1145/1370000/1364786/p9-hayes.pdf?ip=187.122.235.48&id=1364786&acc=OPEN&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E6D218144511F3437&CFID=581778107&CFTOKEN=62156660&__acm__=1455055167_b3b96f9d4414fddc3d82e64d16b48db0 >. Acesso em: 18 de março de 2016.

OLIVEIRA, R.R. Criptografia simétrica e assimétrica: os principais algoritmos de cifragem, 2006. Disponível em: <http://www.ronielton.eti.br/publicacoes/artigorevistaseguran cadigital2012.pdf >. Acesso em: 18 de março de 2016.

CRIPTOGRAFIA ASSIMÉTRICA. Criptografia Assimétrica. 2007. Disponível em: < http://www.gta.ufrj.br/grad/07_2/delio/Criptografiaassimtrica.html >. Acesso em: 18 de março de 2016.

SHANNON, C. Difusão e confusão. 1949 citado em 2002. Disponível em: <http://wiki.di.uminho.pt/twiki/pub/Education/Criptografia/CriptografiaMestrados0405/blocos.pdf >. Acesso em: 20 de agosto de 2016.

STALLINGS, W. *Criptografia e Segurança de Redes:* Princípios e práticas. Ed. Pearson, 2008. 4ª Edição.

LORIN, Sylvain. 2016. PGP-Pretty Good Privacy. Disponível em: < http://br.ccm.net/contents/134-pgp-pretty-good-privacy >. Acesso em: 10 de outubro de 2016.

PGP. How PGP Works. Disponível em: < http://www.pgpi.org/doc/pgpintro/ >. Acesso em: 29 de setembro de 2016.

Introdução à Criptografia e PGP. 2012. Disponível em: < https://memoria.rnp.br/cais/keyserver/ >. Acesso em: 06 de outubro de 2016.

LITTERIO, F. Pretty Good Privacy (PGP), 1997. Disponível em: < http://www.dca.fee.unicamp.br/pgp/pgp.shtml >. Acesso em: 06 de outubro de 2016.

[Callas et al.] CALLAS, J; DONNERHACK, L; FINNEY, H.; SHAW, D.; THAYER, R. Open PGP Message Format. 2007. Disponível em: < https://tools.ietf.org/html/rfc4880 >. Acesso em: 13 de abril de 2017.

BISWAS, P. Content Level Access Control for OpenStack Swift Storage. 2014. Disponível em: < http://profsandhu.com/confrnc/misconf/p123-biswas.pdf >. Acesso em: 20 de agosto de 2016.

NETO, C.C; DE ALMEIDA, M.C; RAPOSO, L.O. SMTP X POP3, TCP X UDP, FTP, HTPP. 2014. Disponível em: <http://www.revista.universo.edu.br/index.php?journal=1reta2&page=article&op=view&path%5B%5D=1158&path%5B%5D=865>. Acesso em: 30 de janeiro de 2017.

Amazon. Disponível em: <https://aws.amazon.com/>. Acesso em: 20 de agosto de 2016.

AMAZON S3. Disponível em: <https://aws.amazon.com/pt/s3/details/>. Acesso em: 18 de março de 2016.

AMAZON S3. Disponível em: < https://www.amazonaws.cn/en/s3/>. Acesso em:18 de março de 2016.

MOIA, V.H.G.; HENRIQUES, M.A.A.; Relação custo/benefício de técnicas utilizadas para prover privacidade em computação nas nuvens. 2014. Disponível em: < http://www.lbd.dcc.ufmg.br/colecoes/sbseg/2014/0028.pdf >. Acesso em: 18 de março de 2016.

STACKSYNC. Stacksync. Disponível em: <http://stacksync.org/>. Acesso em: 18 de março de 2016.

STACKSYNC. Architecture Overview. Disponível em: < http://stacksync.org/architecture-overview >. Acesso em: 18 de março de 2016.

STACKSYNC. Desktop Client. Disponível em: < http://stacksync.org/desktop-client>. Acesso em: 18 de março de 2016.

STACKSYNC. Synchronization Service. Disponível em: <http://stacksync.org/synchronization-service>. Acesso em: 18 de março de 2016.

StackSync: A Dropbox like personal cloud for OpenStack Swift. Speakers: MARTINEZ, A.M.; LOPEZ, P.G.; GONZALEZ, C.C. 2014. Disponível em: <https://www.openstack.org/summit/openstack-summit-atlanta-2014/session-videos/presentation/stacksync-a-dropbox-like-personal-cloud-for-openstack-swift >. Acesso em: 06 de outubro de 2016.

PYDIO. Pydio documentation library. 2012. Disponível em: < **https://pydio.com/en/docs** >. Acesso em: 18 de março de 2016.

_____. Web, desktop and mobile. Disponível em: < https://pydio.com/en/products/web-desktop-and-mobile >. Acesso em: 18 de março de 2016.

_____. Discover Pydio. 2014. Disponível em < http://pt.slideshare.net/Pydio/discover-pydio >. Acesso em: 18 de março de 2016.

OWNCLOUD. OwnCloud 9.0 user manual. Disponível em: < https://doc.owncloud.org/server/9.0/user_manual/files/deleted_file_management.html >. Acesso em: 18 de março de 2016

_____. Disponível em: <https://owncloud.org/> Acesso em: 18 de março de 2016

_____. Cloud Security & Encryption 2.0 Technical Overview. 2015. Disponível em < http://pt.slideshare.net/ownCloud/cloud-security-encryption-20-technical-overview >. Acesso em: 18 de março de 2016.

_____. OwnCloud 5. 2013. Disponível em < http://pt.slideshare.net/wskoczen/own-cloud5 >. Acesso em: 20 de agosto de 2016.

RACKSPACE. Disponível em: <https://www.rackspace.com/pt-br>. Acesso em: 18 de março de 2016.

OPENSTACK SWIFT. Disponível em: < https://www.openstack.org/ >. Acesso em: 18 de março de 2016.

_____. OpenStack Swift. Disponível em: < https://www.swiftstack.com/product/openstack-swift >. Acesso em: 18 de março de 2016.

FTP. O que é FTP. 2016. Disponível em: < http://br.ccm.net/faq/1983-o-que-e-ftp >. Acesso em: 20 de agosto de 2016.

_____. O que é FTP e como usar. 2012. Disponível em: < http://www.techtudo.com.br/artigos/noticia/2012/07/o-que-ftp-e-como-usar.html >. Acesso em: 20 de agosto de 2016.

POSTEL; REYNOLDS. FTP (File Transfer Protocol). 1985. Disponível em: < https://tools.ietf.org/html/rfc959 >. Acesso em: 14 de abril de 2017.

BURNETT, S.; PAINE, S. Criptografia e segurança: O Guia Oficial RSA. 2002. Disponível em: <https://books.google.com.br/books?id=DIskNhcTUNoC&pg=PA58&lpg=PA58&dq=import%C3%A2ncia+do+armazenamento+de+chaves+criptogr%C3%A1ficas&source=bl&ots=H-pyTJMpCg&sig=aQcUB6a88Q-JH8L9bNA3UKVJVT4&hl=pt-BR&sa=X&ved=0ahUKEwje0Z7dv9TLAhUHQZAKHZvtBLAQ6AEIRzAH#v=onepage&q=import%C3%A2ncia%20do%20armazenamento%20de%20chaves%20criptogr%C3%A1ficas&f=false >. Acesso em: 18 de março de 2016.

BERTOL, V.; MENKE, F. Nota da ITI sobre o uso de HSM. 2012. Disponível em: < https://cryptoid.com.br/banco-de-noticias/nota-do-iti-sobre-uso-de-hsm-por-viviane-bertol-e-fabiano-menke/ >. Acesso em: 18 de março de 2016.

MULLER, E.J. A importância de gerar e manter logs. 2014. Disponível em: < http://www.ezequieljuliano.com.br/?p=76 >. Acesso em: 18 de março de 2016.

PYDIO X OWNCLOUD. Pydio x OwnCloud: Comparisons and reviews. Disponível em: <http://www.qoncious.com/questions/pydio-vs-owncloud-comparison-and-review >. Acesso em: 18 de março de 2016.

Evite vulnerabilidades e ameaças na nuvem. 2012. Disponível em: < https://www.ibm.com/developerworks/br/cloud/library/cl-cloudthreats/ >. Acesso em: 06 de outubro de 2016.

FLANAGAN, D. JavaScript: O guia definitivo. 2004. O'Reilly Media Inc.

BROWSERIFY. Browserify Disponível em <http://browserify.org/>. Acesso em: 06 de outubro de 2016.

SOMMERVILLE, I. 2011. Software Engineering. Ed. Pearson. 4 edição.

MDN. Mozzilla Developer Network. Referência da API FileReader. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/API/FileReader>. Acesso em: 13 de abril de 2017.

MOREIRA, Rodrigo. Certificados Digitais. Disponível em: <https://www.gta.ufrj.br/grad/00_1/rodrigo/fr9right.htm>. Acessado em 14 de abril de 2017