UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JUNIOR CESAR DE OLIVEIRA

# UMA ABORDAGEM PARA MANTER CONFIDENCIALIDADE DOS DADOS EM NUVENS DE ARMAZENAMENTO

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO
2014

JUNIOR CESAR DE OLIVEIRA

# UMA ABORDAGEM PARA MANTER CONFIDENCIALIDADE DOS DADOS EM NUVENS DE ARMAZENAMENTO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná – UTFPR.

Orientador: Prof. Me. Luiz Arthur Feitosa dos Santos

CAMPO MOURÃO
2014

# RESUMO

OLIVEIRA, Junior Cesar de. **Uma abordagem para manter confidencialidade dos dados em nuvens de armazenamento**. 2014. 143 f. Trabalho de Conclusão de Curso (Graduação) – Curso Superior de Tecnologia em Sistemas para Internet. Universidade Tecnológica Federal do Paraná. Campo Mourão, 2014.

Devido a alta utilização de tecnologia da informação, uma grande quantidade de dados são gerados diariamente. A computação em nuvem surge para manter essas informações com alta disponibilidade e segurança. No entanto, existem problemas relacionados a confidencialidade das informações armazenadas em ambientes de nuvem de armazenamento, como acessos não autorizados às informações dos usuários e a falta do uso de criptografia nos arquivos armazenados na nuvem. Caso as mesmas não estiverem devidamente protegidas, podem ser violadas. Para minimizar a falta de confidencialidade em nuvens de armazenamento são necessários sistemas capazes de criptografar as informações dos usuários antes que sejam enviadas para a nuvem de armazenamento. Neste trabalho foi proposto uma abordagem que utiliza técnicas de criptografia para manter arquivos de forma segura nos servidores de armazenamento em nuvem. Isso é possível através da implementação de criptografia nos arquivos dos usuários, antes que os arquivos sejam enviados para a nuvem de armazenamento. Na abordagem proposta o próprio usuário do serviço mantém a chave criptográfica, que é utilizada durante o processo de criptografia e descriptografia através de um aplicativo móvel, visando aumentar o nível de segurança dos dados. Dessa forma, a aplicação prove melhorias em relação à confidencialidade das informações armazenadas na nuvem.

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

## ABSTRACT

OLIVEIRA, Junior Cesar de. **An approach to maintain data confidentiality in storage cloud.** 2014. 143 f. Trabalho de Conclusão de Curso (Graduação) – Curso Superior de Tecnologia em Sistemas para Internet. Universidade Tecnológica Federal do Paraná. Campo Mourão, 2014.

Due to high utilization of information technology, a large amount of information are generated daily. Cloud computing emerges to keep this information with high availability and security. However, there are problems related to confidentiality of information stored in cloud storage environments, such as unauthorized access to user information and lack of use encryption in files stored in the cloud. Case the same are not properly protected, can be violated. To minimize the lack of confidentiality in cloud storage are necessary systems able to encrypt the user information before being sent to the storage cloud. In this work we proposed an approach that uses encryption techniques to keep files safely in the cloud storage servers. This is possible through the implementation of user file encryptions, before the files to be uploaded to the storage cloud. In the approach the user himself of service keeps the cryptographic key, which is used during the process of encryption and decryption through a mobile application, order to increase the level of data security. Thus , the application provides improvements in the confidentiality of the information stored in the cloud.



**Keywords**: Cloud Computing, storage cloud, confidentiality, cryptography, security.

# SUMÁRIO

# 1 INTRODUÇÃO

Com a evolução tecnológica, as empresas, unidades governamentais e universidades, passaram a depender cada vez mais da tecnologia da informação. Com isso, o volume das informações aumentam constantemente, e para gerenciá-las são necessários ambientes computacionais capazes de manter as informações sempre disponíveis e seguras. Com o crescimento dos ambientes computacionais, o gerenciamento pode se tornar uma tarefa complexa, pois é necessário que o administrador do ambiente mantenha frequentemente o monitoramento da distribuição dos recursos computacionais do ambiente. Surge então a computação em nuvem, que possibilita o acesso, de modo conveniente e sob demanda, para um conjunto de recursos computacionais configuráveis, como servidores, armazenamento, aplicações e serviços, que podem ser consumidos de forma que os recursos computacionais sejam liberados de acordo com a demanda, fazendo com que haja o mínimo de esforço ou interação com o provedor dos serviços da nuvem (MELL; GRANCE, 2011).

Computação em nuvem é um modelo de computação, no qual os recursos utilizados pelos consumidores estão localizados em uma infraestrutura computacional altamente disponível. Mas, em contrapartida, uma arquitetura muitas vezes nebulosa, pois há um desconhecimento sobre quem pode acessar e gerenciar a nuvem e, consequentemente os dados armazenados nelas. A computação em nuvem pode ser considerada altamente disponível e escalável, pelo fato do consumidor da nuvem ter acesso ao ambiente através da Internet e por seus recursos computacionais serem realocados de acordo com a demanda de algum recurso que o consumidor necessita, como processamento e armazenamento.

O armazenamento de dados é um dos recursos que podem ser disponibilizados pela computação em nuvem, no qual o armazenamento em nuvem prove a entrega de armazenamento de dados por demanda de forma escalável e com suporte a muitos usuários (JONES, 2011). Porém, a alta disponibilidade do armazenamento em nuvem, pode trazer problemas relacionados à segurança, caso as informações armazenadas na nuvem não estejam criptografadas. Dessa forma, as informações armazenadas na nuvem podem ser violadas por entidades governamentais, pela empresa que mantém a nuvem,

ou pessoas maliciosas, caso seja obtido o acesso não autorizado ao ambiente de armazenamento em nuvem. O caso da Agência de Segurança Nacional (NSA) Americana, mostra que as informações dos usuários podem ser violadas antes mesmo dessas informações chegarem ao ambiente de armazenamento em nuvem, devido a criação de um programa de espionagem sobre as informações de usuários de serviços *online* (EFF, 2013).

Assim, há necessidade de utilizar sistemas mais seguros para manter a confidencialidade das informações do consumidor do armazenamento em nuvem. Desta forma, neste trabalho foram desenvolvidas duas ferramentas denominadas AntiNSA e KeyManager, essas ferramentas utilizam uma abordagem para manter a confidencialidade dos dados em nuvens de armazenamento, no qual a ferramenta AntiNSA é responsável por sincronizar arquivos da máquina do usuário com a nuvem de armazenamento e o aplicativo para dispositivos móveis denominado KeyManager, que realiza o gerenciamento das chaves criptográficas utilizadas pela ferramenta AntiNSA, que é o grande diferencial da arquitetura proposta. O objetivo é amenizar o problema da falta de confidencialidade em armazenamento na nuvem utilizando técnicas de criptografia nos arquivos dos usuários antes que sejam enviados para nuvem, de forma que somente os usuários tenham acesso as suas informações.

## 1.1 MOTIVAÇÃO

Os serviços de armazenamento na nuvem, geralmente são disponibilizados por terceiros, assim a responsabilidade de manter as informações dos usuários sempre disponíveis e seguras é uma tarefa da empresa que disponibiliza o serviço, mas algumas das soluções disponíveis não fornecem recursos adequados de segurança (ARAUJO, 2011). O principal problema está relacionado à confidencialidade das informações armazenadas na nuvem. A falta de confidencialidade dos ambientes de nuvem ocorre por fatores como:

- Alguns serviços armazenam seus arquivos sem um mecanismo de criptografia, ou implementam criptografia dentro dos servidores, com isso qualquer pessoa que

tenha acesso ao ambiente de armazenamento pode obter as informações confidencias dos usuários do serviço. Um dos grandes problemas da segurança da informação acontece pelo acesso indevido às informações por funcionários insatisfeitos (PEIXOTO, 2006);

- Como não se sabe onde os dados estão localizados fisicamente, também não se sabe quem tem acesso às informações que podem ser acessados por pessoas não autorizadas;
- O programa de monitoramento da NSA que intercepta informações que trafegam na rede mundial de computadores e posteriormente são enviados à própria NSA (EFF, 2013).
- O fato das nuvens de armazenamento possuírem uma quantidade relevante de informações dos usuários, com isso, pode haver uma concentração de ataques devido ao interesse aos dados (CASTRO et al., 2012).

A falta de confidencialidade nos ambientes de armazenamento nas nuvens é uma das desvantagens dessa tecnologia, pois gera um certo desconforto e desconfiança dos usuários que utilizam o serviço, fazendo com que esse tipo de serviço não seja adotado por usuários que possuem informações confidenciais.

## 1.2 JUSTIFICATIVA

A justificativa para o desenvolvimento desse trabalho é porque há problemas de confidencialidade com as informações armazenadas nas nuvens (ROCHA, 2010). Esse problema pode ser minimizado nos serviços de armazenamento em nuvem, porém algumas ferramentas de armazenamento em nuvem não implementam criptografia nos dados antes de serem enviados à nuvem (SILVA, 2012), e também há um desconhecimento ou falta de confiança sobre como é feito o tratamento da confidencialidade em alguns serviços de armazenamento nas nuvens, pois a arquitetura de software muitas vezes não é divulgada.

## 1.3 OBJETIVO

Neste trabalho objetiva-se analisar problemas relacionados a confidencialidade em ambientes de computação em nuvem e desenvolver uma ferramenta que proporcione melhorias na segurança aos dados armazenados nas nuvens.

Portanto, para minimizar os problemas relacionados à falta de confidencialidade em nuvens de armazenamento, foi proposto uma arquitetura para manter os dados dos usuários de serviços de armazenamento em nuvem seguros, utilizando técnicas de criptografia nos dados dos usuários, antes que os dados sejam enviados para à nuvem de armazenamento. Para prova de conceito foi desenvolvido um protótipo de software que implementa a arquitetura proposta.

O presente trabalho esta estruturado da seguinte maneira, o Capítulo 2 apresenta o referencial teórico; o Capítulo 3 apresenta os trabalhos relacionados; o Capítulo 4 apresenta a metodologia, a arquitetura e a ferramenta desenvolvida; e por fim a conclusão e trabalhos futuros.

# 2 REFERÊNCIAL TEÓRICO

Nesta seção são explicados alguns conceitos necessários para o entendimento do trabalho proposto, no qual são apresentadas algumas características da computação em nuvem e seus modelos de serviços e implantação, também são abordados os modelos de implantação de armazenamento em nuvem e seu funcionamento.

## 2.1 COMPUTAÇÃO EM NUVEM

O termo computação em nuvem está associado a um novo paradigma de computação, no qual essa tecnologia tende a deslocar a infraestrutura local para uma infraestrutura disponibilizada através da Internet. Assim, os custos de software e hardware podem ser consideravelmente reduzidos, pelo fato do consumidor da nuvem pagar somente pelos recursos utilizados. Apesar do termo computação em nuvem ser muito discutido, ainda não há uma definição clara e completa na literatura sobre o assunto (VAQUERO et al., 2009).

Porém, no trabalho de Vaquero et al. (2009) foi elaborada uma definição para computação em nuvem que contempla três conceitos: virtualização, que é a criação de ambientes virtuais para os usuários, omitindo as características físicas do ambiente computacional; escalabilidade, ou seja, a capacidade de aumento ou redução de um ambiente virtual; e o modelo *pay-per-use,* no qual o usuário paga somente pelo consumo do recurso utilizado. No entanto, o uso de virtualização na nuvem não é obrigatório, pois é possível utilizar recursos da nuvem somente com máquinas físicas e se algum utilizador implementar sua própria nuvem para um ambiente privado, o modelo *pay-per-use* não se encaixará nesse tipo de ambiente, pelo fato do ambiente pertencer ao utilizador.

A computação em nuvem pode ser definida da seguinte maneira:

> “Computação em nuvem é um modelo que possibilita acesso à rede de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser

> rapidamente adquiridos e liberados com o mínimo de esforço ou interação com o provedor dos serviços.” (MELL; GRANCE, 2011).

Para detalhar melhor esse assunto, é apresentada a composição do modelo de computação em nuvem de acordo com o trabalho de MELL e GRANCE (2011). O modelo de computação em nuvem é composto por cinco características básicas, três modelos de serviços e quatro modelos de implementação. Essas características e modelos de computação em nuvem são detalhadas respectivamente nas três seções a seguir.

### 2.1.1 Características básicas

A seguir são apresentadas as características básicas de um ambiente de computação em nuvem, de acordo com a abordagem de MELL e GRANCE (2011):

- *Self-Service* sob demanda: o usuário pode consumir unilateralmente recursos computacionais, como tempo de processamento no servidor e armazenamento em rede, conforme o necessário de forma que não haja necessidade de interação humana com os provedores de cada serviço;
- Amplo acesso a rede: os recursos estão disponíveis através da rede e são acessados pelos consumidores dos recursos por vários tipos de dispositivos como por exemplo, *notebooks*, *tablets* e celulares;
- *Polling* de recursos: o provedor dos recursos de computação em nuvem é agrupado para atender vários consumidores através de um modelo *multi-tenant*[1], com diferentes recursos físicos e virtuais atribuídos de acordo com a demanda do consumidor de forma dinâmica. No qual, o cliente geralmente não tem nenhum controle ou conhecimento sobre a localização exata dos recursos disponibilizados. Pode ser citado como exemplo de recursos o armazenamento, o processamento, a largura de banda de rede e as máquinas virtuais;
- Elasticidade rápida: os recursos da computação em nuvem podem ser adquiridos de forma rápida e elástica, em alguns casos os recursos são

1 Nesse modelo uma aplicação é projetada para particionar seus dados e configurações, e cada cliente trabalha com uma instância virtual do aplicativo.

liberados de acordo com o aumento da demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento;

- Serviço medido: sistemas em nuvem controlam e otimizam automaticamente a utilização dos recursos, aproveitando a capacidade de medição em algum nível de abstração adequado para o tipo de serviço. O uso de recursos pode ser monitorado e controlado, oferecendo transparência para o provedor e o consumidor.

### 2.1.2 Modelos de implantação

A computação em nuvem pode ser utilizada de acordo com a necessidade do utilizador, para isso, existem quatro modelos de implantação que são o modelo de infraestrutura de nuvem pública, o de nuvem privada, o de nuvem híbrida e o de nuvem comunitária, que são detalhados logo a seguir.

A infraestrutura de uma nuvem pública é destinada para o uso ao público em geral, e gerenciada e operada por uma empresa, órgão acadêmico ou governamental, ou até mesmo uma combinação entre eles (MELL; GRANCE, 2011). O modelo de nuvem pública pode ser acessado por qualquer usuário que conheça a localização do serviço, e não podem ser aplicadas técnicas para restringir o acesso quanto ao gerenciamento de redes e menos ainda, aplicar técnicas de autenticação e autorização (MACHADO et al., 2009).

No modelo de implantação de nuvem privada, seu uso é exclusivo para uma organização, no qual a nuvem é administrada pela própria empresa ou por terceiros. Neste modelo de implantação são empregados políticas de acesso aos serviços que podem ser em nível de gerenciamento de redes, configurações dos provedores de serviços e a utilização de tecnologias de autenticação e autorização. Um bom exemplo deste modelo seria um cenário de uma universidade e suas dependências, no qual a universidade gerencia quem poderá utilizar os serviços disponibilizados pelo ambiente de nuvem (MACHADO et al., 2009).

O modelo de nuvem híbrida é uma composição de duas ou mais nuvens diferentes,

que podem ser privadas, comunitária ou pública, e que permanecem como entidades únicas e são ligadas por uma tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações (MELL; GRANCE, 2011).

No modelo de implantação de nuvem comunitária ocorre o compartilhamento por empresas de uma nuvem, sendo esta suportada por uma comunidade específica que partilhou seus interesses, tais como a missão, os requisitos de segurança, política e considerações sobre flexibilidade (MACHADO et al., 2009). O modelo de nuvem comunitária pode ser gerenciado e operados por várias organizações da comunidade, por um terceiro, ou alguma combinação entre eles (MELL; GRANCE, 2011).

### 2.1.3 Modelos de serviços

A computação em nuvem distribui os serviços na forma de recursos, no qual, os modelos de serviços são divididos em três, como mostra a Figura 1.

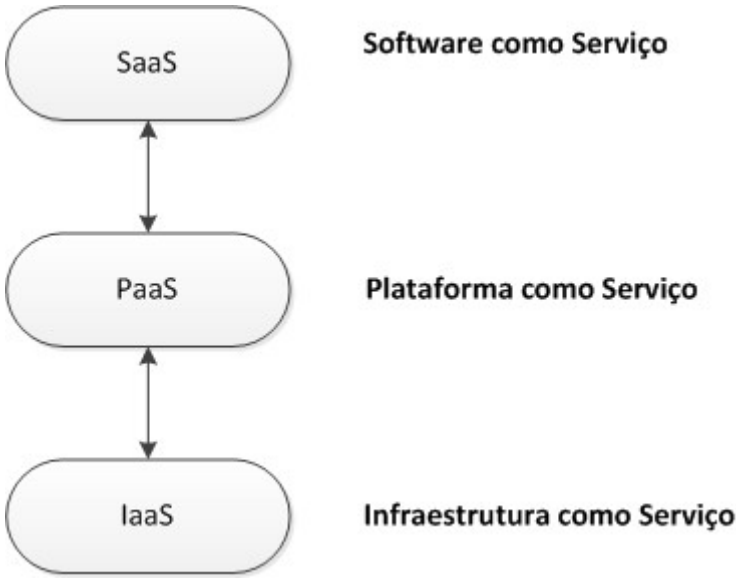


Figura 1 – Modelo de serviços da computação em nuvem
Fonte: Adaptado de Silva (2010).

O modelo Software como um Serviço ou SaaS *(Software as a Service)* representa os serviços de mais alto nível na computação em nuvem, que são aplicações completas fornecidas aos usuários. Os prestadores de serviços disponibilizam o SaaS na camada de

aplicação da nuvem, no qual essas aplicações são executados inteiramente na nuvem e o consumidor não precisa gerenciar ou controlar a infraestrutura. Dessa forma esses serviços podem ser considerados como uma alternativa a executar um programa em uma máquina local, dispensando a instalação da aplicação por parte do consumidor e aquisição de licença de software (SILVA, 2010). Um bom exemplo de um serviço disponibilizado no modelo SaaS é o Google Docs[2], no qual é possível utilizar uma aplicação de edição de documentos, através de um navegador.

O modelo de Plataforma como um Serviço ou PaaS *(Plataform as a Service)* pode facilitar o desenvolvimento de aplicações destinadas aos consumidores. O PaaS fornece uma infraestrutura de alto nível de integração para que desenvolvedores possam implementar e testar aplicações na nuvem, também oferece sistemas operacionais, linguagens de programação e ambientes de desenvolvimento para aplicações. Disponibiliza ferramentas de desenvolvimento e de colaboração entre desenvolvedores para facilitar a implementação de softwares (SILVA, 2010). Um exemplo de PaaS é a ferramenta oferecida pela Google, o Google AppEngine[3] que fornece auxilio ao desenvolvimento e implementação de aplicações, sem ser necessário se preocupar com hardware, cópias de segurança e também dispõe de uma alta escalabilidade.

O modelo Infraestrutura como um Serviço ou IaaS *(Infrastructure as a Service)* é responsável por prover toda infraestrutura necessária para outros modelos de serviço, o SaaS e o PaaS. O objetivo o modelo IaaS é tornar mais fácil e acessível o fornecimento de recursos, como servidores, rede, armazenamento e outros recursos necessários para construir um ambiente de aplicação sob demanda, que podem incluir sistemas operacionais e aplicativos (MELL; GRANCE, 2011).

O termo IaaS se refere a uma infraestrutura computacional baseada em técnicas de virtualização de recursos de computação, no qual os recursos computacionais são alocados de forma dinâmica e de acordo com a necessidade das aplicações. Dessa forma, os recursos computacionais disponíveis podem ser melhor aproveitados, de forma que sejam utilizados grande parte dos recursos disponíveis no ambiente (MACHADO et al., 2009). O Amazon EC2[4] (*Elastic Cloud Computing*) e o Eucalyptus[5] (*Elastic Utility Computing Architecture Linking Your Programs To Useful Systems*) são exemplos de

2 https:/docs.google.com
3 https://appengine.google.com
4 http://aws.amazon.com/ec2/
5 http://www.eucalyptus.com/

modelos IaaS, no qual esses sistemas fazem o gerenciamento dos recursos disponíveis na infraestrutura da nuvem.

## 2.2 ARMAZENAMENTO EM NUVEM

O armazenamento de dados em nuvem é uma das formas de oferta de recursos computacionais da computação em nuvem, no qual o provedor da nuvem fornece acesso ao usuário para que ele possa enviar e obter suas informações por meio de aplicações que implementam interfaces *web services*[6], que permite a comunicação entre aplicações de uma maneira independente de sistema operacional e de linguagem de programação. A Figura 2 apresenta uma visão geral da arquitetura de sistemas de armazenamento em nuvem. O armazenamento de dados nas nuvens está presente dentro do modelo de serviço IaaS (SILVA et al., 2012).

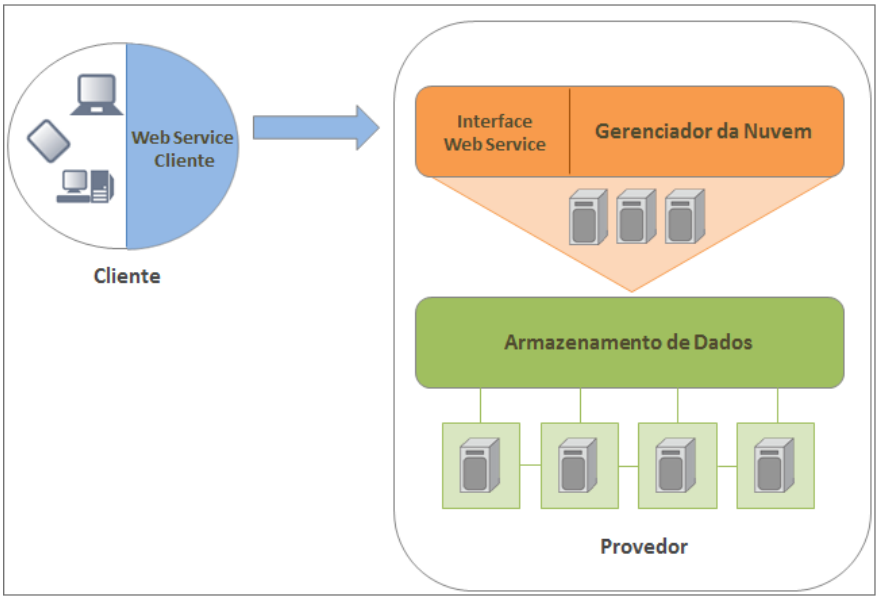


Figura 2 – Arquitetura geral de sistemas de armazenamento em nuvem.
Fonte: SILVA et al. (2012).

O armazenamento em nuvem não se trata somente de um dispositivo de armazenamento, mas sim do uso de todo o sistema de armazenamento em nuvem em um serviço de acesso aos dados. O núcleo do armazenamento em nuvem é composto por

6 Nome dado à tecnologia que permite a comunicação entre aplicações de uma maneira independente de sistema operacional e de linguagem de programação.

aplicações e dispositivos de armazenamento, fazendo com que a aplicação transforme o dispositivo de armazenamento em um serviço de armazenamento em nuvem (ZHU, 2012).

Em geral o armazenamento de dados em nuvem prove a entrega de armazenamento por demanda de forma escalável e com suporte a muitos usuários (JONES, 2011).

No Quadro 1 é possível visualizar a definição de algumas características do armazenamento em nuvem.

| Característica | Descrição |
|---|---|
| Gerenciamento | A capacidade de gerenciar um sistema com o mínimo de recursos |
| Método de acesso | Protocolo através do qual o armazenamento em nuvem é exposto |
| Desempenho | Desempenho medido por largura de banda e latência |
| Multi locação | Suporte para diversos usuários |
| Escalabilidade | Capacidade de ser escalável para atender demandas de recursos computacionais de maneira eficiente |
| Disponibilidade de dados | Medida do tempo de atividade de um sistema |
| Controle | Capacidade de controlar um sistema em particular, para configurar para custo desempenho ou outras características |
| Eficiência de armazenamento | Medida da eficiência com que o armazenamento bruto é usado |
| Custo | Medida do custo do armazenamento (normalmente em dólares por gigabyte) |

Quadro 1 - Características do armazenamento em nuvem.
Fonte: Adaptado de JONES (2011).

De acordo com Jones (2011), os serviços de armazenamento em nuvem são divididos em três modelos, que usam um conceito similar ao da divisão dos modelos de computação em nuvem.

Os modelos de armazenamento em nuvem podem ser vistos na Figura 3. O primeiro modelo mostrado é o armazenamento em nuvem privada, que usa o mesmo

conceito de nuvem de armazenamento pública, porém, de uma forma que possa ser integrados com meios de segurança dentro de um *firewall*[7]. O segundo modelo é armazenamento em nuvem pública que apresenta uma infraestrutura como um serviço para aluguel, ou seja, que possa ser utilizado por qualquer usuário que necessite do serviço de armazenamento em nuvem. E por fim, o modelo de armazenamento em nuvem híbrida, no qual existe uma composição de duas ou mais nuvens, que podem ser privadas, comunitária ou pública, dessa forma é possível definir políticas para selecionar quais dados serão armazenados de forma privada e quais podem ser protegidos em nuvens públicas (JONES, 2011).

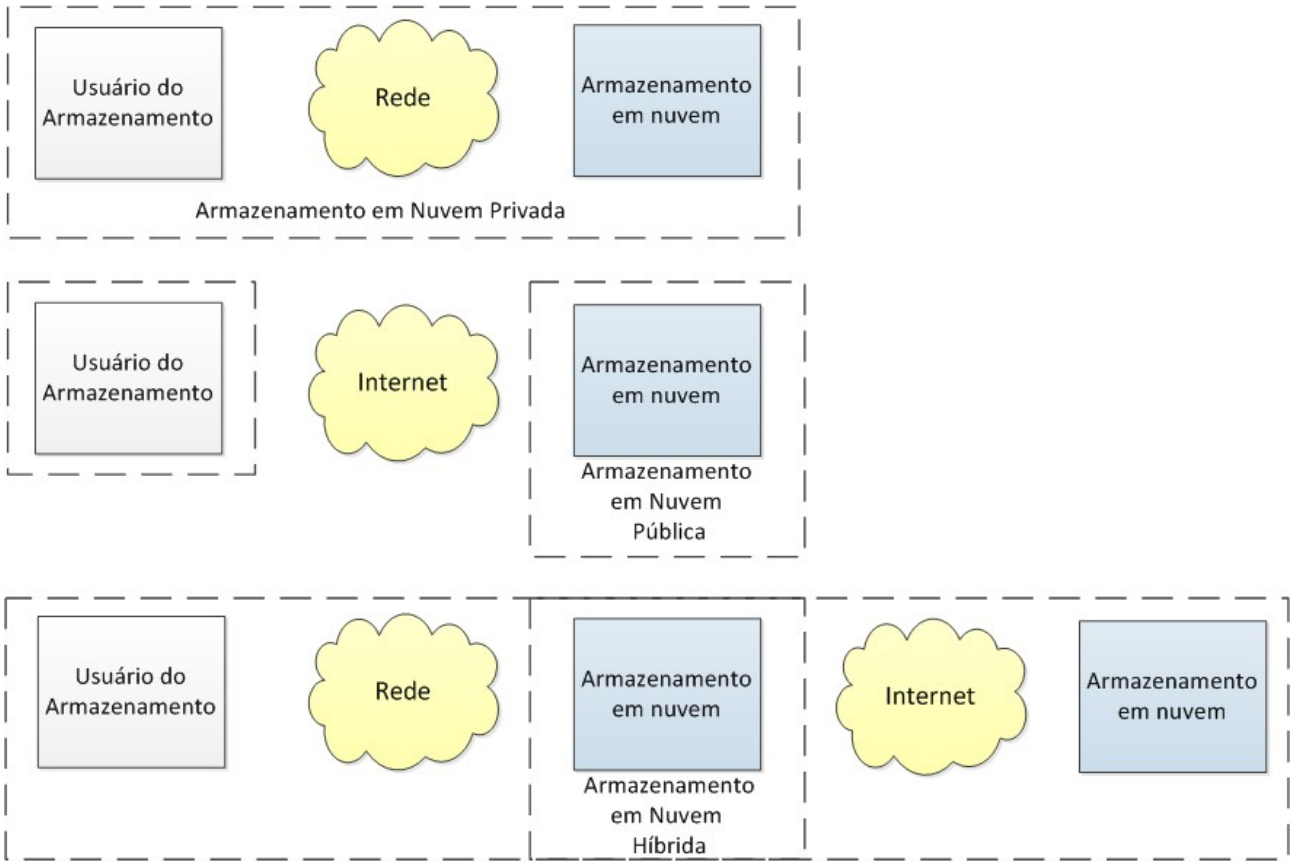


Figura 3 - Modelos de armazenamento em nuvem.
Fonte: JONES (2011).

O armazenamento em nuvem é uma tecnologia que pode ser muito interessante tanto para pessoas, quanto para empresas. Os usuários podem ter suas informações sempre disponíveis em qualquer lugar que possui acesso à Internet e as empresas poderiam utilizar serviços de armazenamento para manter seus dados ou implantar sua própria nuvem privada.

7 Dispositivo de uma rede de computadores capaz de aplicar uma política de segurança em um ponto de rede.

Apesar dos benefícios oferecidos pelo armazenamento em nuvem, como a disponibilidade e escalabilidade, ainda existem problemas relacionados a segurança das informações armazenadas nas nuvens.

Um dos problemas de segurança na nuvem é a falta de confidencialidade que é gerada pelo fato do provedor da nuvem, ter acesso as informações dos usuários que armazenam dados em seus ambientes (ROCHA, 2010). No qual, somente pessoas autorizadas pelo usuário do serviço de armazenamento em nuvem devem ter acesso aos dados. Para isso é necessário a criação de aplicações clientes que sejam capazes de criptografar os dados no próprio dispositivo do usuário, de forma que os dados sejam enviados em uma forma não compreensível para o provedor da nuvem (KUMAR et al., 2012).

No presente trabalho foi realizado o desenvolvimento de uma arquitetura que atenda as necessidades citadas anteriormente. Na Figura 4 é apresentada a arquitetura proposta e que pode melhorar o problema de falta de confidencialidade em serviços de armazenamento em nuvem.

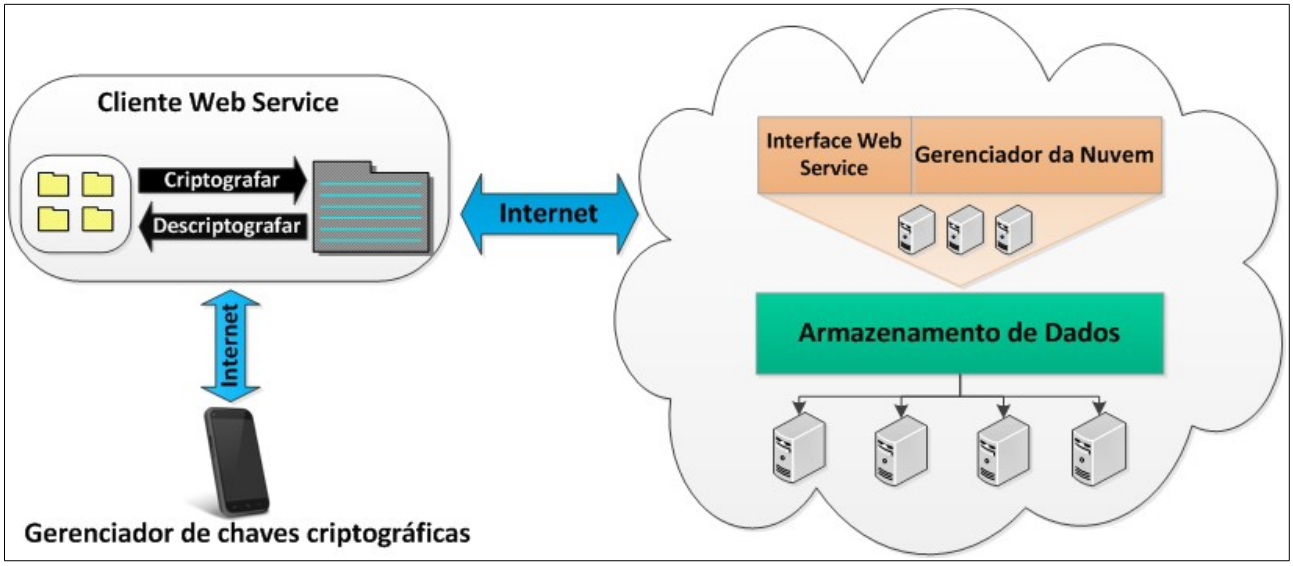


Figura 4 – Abordagem para o cliente para manter dados seguros no armazenamento em nuvem.
Fonte: Adaptado de SILVA et al. (2012).

## 2.3 PROTOCOLO DE AUTORIZAÇÃO OAUTH 2.0

O *framework* de autorização OAuth 2.0 permite que um aplicativo de terceiros

obtenham acesso limitado a um serviço Hypertext Transfer Protocol (HTTP), no qual o processo de obtenção de acesso ao serviço HTTP pode ser autorizado em nome do proprietário do serviço ou em nome de um aplicativo de terceiros (OAUTH, 2012).

No processo de autorização o OAuth permite que o cliente realize uma solicitação de acesso aos serviços ou recursos controlados pelo provedor do serviço. Em seguida o cliente obtém a autorização de acesso e são fornecidas as credenciais para que o serviço possa ser accessado pelo cliente, sem ser necessário trafegar as senhas de acesso ao serviço .

O OAuth define quatro funções para que o processo de autorização possa ser realizado (OAUTH, 2012):

- Proprietário do recurso: uma entidade capaz de possibilitar o acesso a um recurso protegido;
- Servidor de recursos: um servidor que hospeda os recursos protegidos, capaz de aceitar e responder às solicitações de acesso aos recursos utilizando tokens de acesso[8];
- Cliente: uma aplicação que realiza as solicitações aos recursos protegidos;
- Servidor de autorização: um servidor capaz de emitir tokens de acesso para o cliente, após o processo de autenticação e obtenção de autorização.

Por fim, a ferramenta desenvolvida utiliza o serviço de armazenamento do Google Drive[9], que por sua vez utiliza o *framework* de autorização OAuth 2.0 para permitir que aplicações de terceiros possam acessar o recurso de armazenamento em nuvem. Assim, é necessário que o usuário da ferramenta realize o processo de autorização para obter acesso ao recurso de armazenamento disponibilizado pelo Google Drive.

## 2.4 CRIPTOGRAFIA

A criptografia é uma ciência extremamente importante para segurança da informação, pois serve de base para diversas tecnologias e protocolos como a infraestrutura de chaves públicas (PKI), o protocolo de segurança IP (IPSec) e o *Wired*

8 É uma string representando uma autorização concedida ao cliente.
9 https://drive.google.com

*Equivalent Privacy* (WEP). A utilização de criptografia permite que propriedades fundamentais da segurança da informação sejam alcançadas, dentre as quais estão a integridade, autenticidade, confidencialidade e o não-repúdio. E devido a alta utilização da comunicação pelas pessoas, é necessário que a segurança dessa comunicação seja sempre garantida para manter a privacidade dos usuários (NAKAMURA; GEUS, 2007).

O grande objetivo da criptografia é transferir uma mensagem ou arquivo, chamado de texto claro, que pode ser entendido pelo usuário, e criptografá-lo em um texto cifrado de modo que somente as pessoas autorizadas saibam como transformá-lo em sua forma original novamente (TANENBAUN, 2009).

### 2.4.1 Criptografia de chave simétrica

A criptografia de chave simétrica, também conhecida como criptografia de chave única, consiste no uso de uma única chave para cifrar e decifrar os dados(STALLINGS, 2008).

De acordo com Stallings (2008), um esquema de criptografia simétrica é composto por cinco elementos que são:

- Texto claro: contém a mensagem original, que pode ser compreendida pelo usuário e é utilizada como entrada para o algoritmo de criptografia;
- Algoritmo de criptografia: realiza diversas substituições e transformações no texto claro para torná-lo ilegível;
- Chave secreta: utilizada como entrada para o algoritmo de criptografia, no qual a chave é um valor independente do texto claro e do algoritmo. Assim, o algoritmo produzirá saídas diferentes, de acordo com a chave utilizada;
- Texto cifrado: mensagem produzida como saída, que depende do texto claro e da chave secreta.
- Algoritmo de descriptografia: é o algoritmo de criptografia executado de modo inverso, fazendo o uso da chave secreta e o texto cifrado para obter o texto claro original.

Os algoritmos de chave simétrica tem como característica a rapidez na execução,

porém há dificuldades para a distribuição das chaves secretas utilizadas pelos usuários de maneira segura, devido à falta de segurança dos canais de comunicação. (NAKAMURA; GEUS, 2007).

Por fim, a criptografia é um dos mecanismos utilizados para fornecer segurança aos dados das pessoas, devido ao fato de atender as propriedades básicas da segurança da informação. O presente trabalho utiliza a criptografia para fornecer confidencialidade nos sistemas de armazenamento em nuvem, no qual foi utilizado o algoritmo de criptografia de chave simétrica *Advanced Encryption Standard* (AES), pois o mesmo é considerado eficiente quando se trata de processamento e pelo fato da arquitetura não fornecer meios para o compartilhamento de arquivos, assim não há necessidade de utilizar o compartilhamento de chaves.

## 2.5 PRIVACIDADE DOS SERVIÇOS *ONLINE*

Um programa de espionagem da NSA foi implementado pelo presidente George W. Bush, após os ataques de 11 de setembro, com objetivo de combater o terrorismo, porém esse programa não visava somente terroristas, mas possivelmente milhares de americanos que nunca foram considerados suspeitos de cometerem crimes (EFF, 2013).

Após obter informações sobre o programa de espionagem, o ex-técnico da Agência Central de Inteligência(CIA) Americana, Edward Snowden, disponibilizou uma série de documentos sigilosos de segurança dos Estados Unidos que revelaram em detalhes alguns programas de vigilância que utilizam servidores de empresas como Google, Apple e Facebook para espionar a população Americana e vários países da Europa e da América Latina, inclusive o Brasil. Snowden obteve acesso aos documentos enquanto prestava serviços para a NSA, pois ele decidiu que não poderia deixar que o governo dos Estados Unidos destruísse a privacidade, a liberdade da Internet e os direitos básicos de pessoas em todo mundo (G1, 2014).

Snowden afirmou que a NSA espionava por motivos comerciais, citando um exemplo sobre a empresa Siemens[10]. Pois, quando existem informações sobre a

10 http://www.siemens.com/

Siemens, que sejam de interesse dos Estados Unidos, porém não tenham nenhuma relação com a segurança nacional do país, as informações seriam coletadas da mesma maneira (WELLE, 2014).

Segundo o jornal The Washington Post[11] a NSA invadiu em segredo canais de comunicação que conectam *data centers* do Yahoo e Google ao redor do mundo, dessa forma, a NSA obtém acesso a dados de centenas de milhares de contas de usuários dos serviços da Yahoo e do Google (GREENWALD; MACASKILL, 2014). Os acessos realizados pela NSA fazem parte de um programa chamado PRISM, que permite que a NSA obtenha materiais como o histórico de pesquisa, conteúdo de e-mails, transferências de arquivos e *chats* ao vivo. Essas informações foram obtidas através de um documento secreto obtido pelo Guardian[12], que verificou a autenticidade do documento considerado ultrassecreto. Apesar do documento conter afirmações de que o programa é executado com o apoio das empresas, algumas empresas como a Google e Apple disseram que não teriam conhecimento do programa de vigilância realizado pela *NSA* (GREENWALD; MACASKILL, 2014).

Para realizar a espionagem, o governo Americano convenceu grandes empresas de telecomunicações dos EUA como AT&T[13], MCI[14] e Sprint[15]. Em seguida, essas empresas permitiram que a NSA instalasse equipamentos vigilância das comunicações muito sofisticados, dessa forma, a NSA obteve acesso irrestrito a fluxos de dados nacionais e internacionais em tempo real diariamente. Os equipamentos utilizados para realizar análises são capazes de analisar dezenas de *gigabits* de pacotes de redes em segundos, e dessa forma eram realizadas cópias exatas dos dados que trafegavam através das linhas de telecomunicações que eram violadas, e enquanto um fluxo dos dados era redirecionado para o governo, o outro seguia para os seus devidos destinos (EFF, 2013).

O programa PRISM permite que a NSA obtenha comunicações direcionadas sem ter de solicitar aos fornecedores de serviços e sem a necessidade de obter ordens judiciais individuais. Isso foi permitido através de mudanças na lei de vigilância dos EUA, introduzidas pelo presidente George Bush e renovada por Barack Obama.

11 http://www.washingtonpost.com/
12 http://www.theguardian.com/
13 http://www.att.com/
14 http://www.mci.com/
15 http://www.sprint.com/

(GREENWALD; MACASKILL, 2014).

A Figura 5 exibe um dos gráficos que estão contidos no documento ultrassecreto obtido pelo Guardian, o gráfico exibe os dados que o programa de espionagem da NSA é capaz de violar, entre os quais aparecem grandes empresas de tecnologia, como Google, Microsoft e Apple.

Com o conhecimento do programa de espionagem do governo americano, os usuários de serviços *online* passaram a questionar a questão da privacidade das informações armazenadas nas nuvens. Assim, a confidencialidade dos serviços se torna um fator extremamente importante no momento em que um usuário resolve adquirir algum serviço *online* para armazenar suas informações confidenciais, sejam elas pessoais ou comerciais.

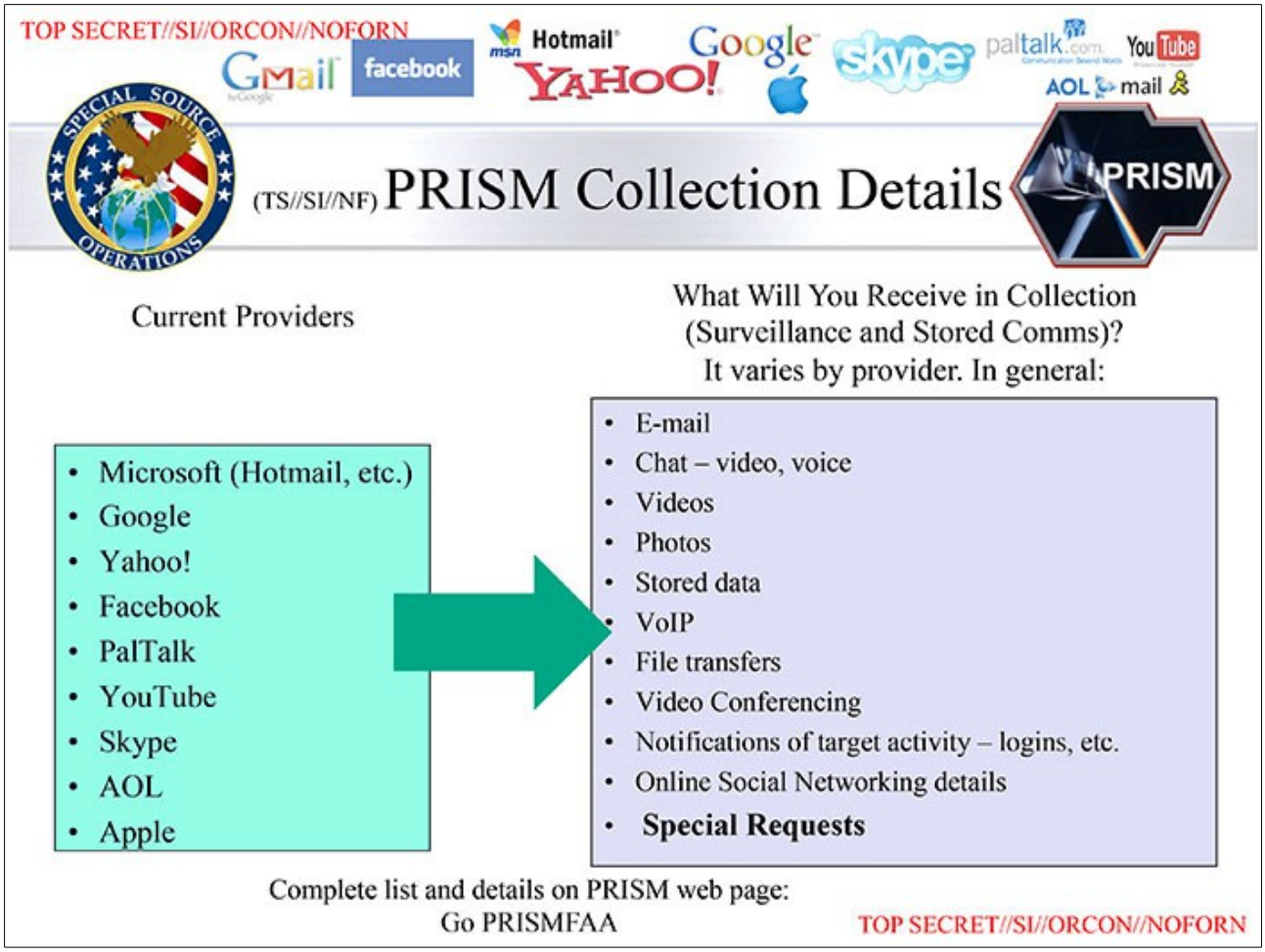

Figura 5 – Detalhes da coleta de dados do programa PRISM.
Fonte: THE WASHINGTON POST (2014).

Portanto, pode ser visualizado que os provedores de serviços *online* não tem total controle de acesso às informações armazenadas em seus servidores, devido ao fato de

que as informações podem ser interceptadas, antes mesmo se serem enviadas ao destino. Esse problema pode ser minimizado, caso as informações sejam criptografadas antes de serem enviadas ao destino, e mesmo que fossem interceptadas, teriam de ser descriptografadas para se tornarem legíveis. Dessa forma o trabalho dos sistemas que interceptam essas informações seria maior, pois haveria a necessidade de descobrir as chaves para descriptografar as informações que trafegam na rede mundial de computadores.

# 3 TRABALHOS RELACIONADOS

Nesta seção são apresentados os trabalhos relacionados com a confidencialidade de dados: Rocha (2010) mostra como um administrador malicioso pode acessar um ambiente de nuvem e obter as informações dos usuários; Silva et al. (2012) apresentam alguns aspectos de segurança necessários para que uma nuvem de armazenamento seja segura e informa que existe um sistema de armazenamento de código aberto que não possui aplicações clientes seguras para manter os dados nas nuvem; Kumar et al. (2012) propõem um método para manter os dados dos usuários do armazenamento em nuvem seguros; e por fim Chen et al. (2013) apresentam um sistema chamado SecureDropbox, que fornece criptografia nos arquivos armazenados na nuvem.

## 3.1 PRIVACY IN CLOUD COMPUTING

Rocha (2010) afirma que o fato dos provedores de computação em nuvem terem controle total sobre os dados de seus usuários pode influenciar muito na decisão de migração para utilizar o ambiente de nuvem.

De acordo com Rocha (2010), caso um administrador malicioso tenha acesso total ao ambiente da nuvem, o administrador tem a possibilidade de aplicar várias técnicas de ataque e fazer o uso de ferramentas que podem comprometer a segurança dos dados dos usuários armazenados na nuvem, ou seja, as informações dos usuários terão sua privacidade comprometida.

Para realizar o trabalho, Rocha (2010) montou uma infraestrutura de nuvem, semelhante as que são encontradas no mercado, porém, em uma escala menor. Em seu trabalho, Rocha (2010) não considera sistemas de armazenamento que implementam criptografia no cliente, pois o objetivo é mostrar como um administrador malicioso pode comprometer a segurança de dados que são utilizados dentro da nuvem, isso demonstra que pode existir ambientes sem proteção, pois os ambientes deveriam impedir qualquer tipo de acesso não autorizado nas informações dos usuários.

Para demonstrar como o administrador malicioso pode obter acesso aos dados confidencias de usuários, Rocha (2010) descreve um ataque aos discos rígidos em um ambiente de nuvem, no caso o administrador da nuvem, pode acessar os discos rígidos e realizar cópias das informações sem passar pelo controle de privilégios da vítima.

As informações sobre como um administrador malicioso pode comprometer um ambiente de computação em nuvem, servem de motivação para que sejam estudados novos meios de obter a segurança e privacidade das informações dos usuários de computação em nuvem, tal como propõem o presente trabalho.

O fato do trabalho de Rocha (2010) não levar em consideração sistemas de armazenamento nas nuvens que dispõe de criptografia no cliente, mostra que esse tipo de abordagem pode ser uma solução um pouco mais confiável para os usuários que necessitam utilizar as nuvens de armazenamento para manter seus dados confidenciais. Dessa forma, há uma necessidade de criação de aplicações clientes seguras para nuvens de armazenamento que não fornecem essa proteção aos dados dos usuários, tal como ao protótipo desenvolvido por nosso trabalho.

## 3.2 ASPECTOS DE SEGURANÇA EM NUVENS DE ARMAZENAMENTO

O trabalho de Silva et al. (2012) expõe que muitos provedores de nuvens de armazenamento afirmam que seus ambientes são mais seguros do que nos dispositivos de seus usuários. Porém, nem sempre essas nuvens seguem todos critérios de segurança, como criptografia dos dados e comunicação segura, tornando as informações vulneráveis a diversos tipos de ataques, o que pode comprometer a confidencialidade dos dados dos usuários.

De acordo com Silva et al. (2012), o ambiente de nuvem apresenta várias fragilidades devido às suas características, como a alta disponibilidade das informações e a capacidade de satisfazer demandas por recursos. Assim, agentes maliciosos visam obter os dados dos usuários que estão armazenados nas nuvens. Devido a esse fato, são apresentados alguns aspectos de segurança para melhorar a segurança em ambientes de nuvem. Os aspectos são: criptografia de dados no cliente; comunicação segura nos

canais de comunicação do serviço; monitoramento de eventos; autenticação; gerência de usuários e criptografia de dados no servidor.

Por fim, foi verificado a utilização dos aspectos de seguranças descritos por Silva et al. (2012), em um dos principais sistemas de código aberto para armazenamento nas nuvens, o OpenStack Swift[16]. O trabalho identificou que o OpenStack Swift está conforme os seguintes aspectos de segurança: a comunicação segura entre cliente e servidor e a utilização de meios de autenticação e autorização de acesso aos dados armazenados. No entanto, a ferramenta não apresenta suporte a criptografia de dados no cliente e não fornece suporte à criptografia no servidor. O que certamente motiva e justifica a proposta do presente trabalho.

Devido ao fato da ferramenta analisada por Silva et al. (2012) não possuir suporte a aplicações clientes seguras e esse aspecto ser considerado necessário para manter a confidencialidade dos dados nas nuvens, no presente trabalho foi realizado o desenvolvimento de duas ferramentas que podem atender essa necessidade.

## 3.3 SECURE STORAGE AND ACCESS OF DATA IN CLOUD COMPUTING

Kumar et al. (2012) propõem um método que permite ao usuário da nuvem, acessar e armazenar os dados de uma forma segura. Kumar et al. (2012) afirma que ninguém, exceto o usuário autenticado tem acesso aos dados armazenados nas nuvens.

De acordo com Kumar et al. (2012) os dados dos usuários podem ser divididos em duas seções na nuvem de armazenamento, no qual uma contém os dados privados e a outra os dados compartilhados. Para que isso seja feito de forma segura, Kumar et al. (2012) conta com a utilização da abordagem *ECC (Eliptic Curve Cryptography ou* Criptografia de Curvas Elípticas), no qual é fornecida uma chave privada para os dados armazenados na seção privada e uma chave pública para seção compartilhada. Kumar et al. (2012) afirma que a abordagem *ECC* é mais eficiente do que os sistemas baseados em *RSA*[17], pelo fato da *ECC* utilizar chaves menores para garantir um nível de segurança

16 http://swift.openstack.org
17 Algoritmo de criptografia de dados, que deve o seu nome a três professores que são Ronald Rivest, Adi Shamir e Leonard Adleman.

equivalente ao *RSA*.

O método proposto por Kumar et al. (2012) funciona da seguinte forma, o usuário realiza autenticação na nuvem de armazenamento para que o prestador de serviços possa carregar o módulo de criptografia para o dispositivo do cliente, em seguida o cliente deve informar um número para que o módulo possa gerar a chave secreta. Por fim, o módulo de criptografia utiliza a chave secreta para criptografar os dados do usuário antes de serem enviados para a nuvem. E quando o usuário deseja acessar os dados, o servidor envia as informações criptografadas para o cliente, em seguida o módulo de criptografia utiliza a mesma chave secreta, que foi usada para criptografar, no processo de decifrar os dados para o estado original.

O presente trabalho pretende utilizar a abordagem de Kumar et al. (2012) para manter os dados seguros na nuvem. A abordagem de manter a chave secreta no cliente e realizar todas operações de criptografar e descriptografar no próprio cliente é uma forma eficaz de assegurar que as informações estão seguras contra acesso não autorizado aos dados feito pelo provedor da nuvem e em caso que haja violação de segurança no ambiente. No entanto, não foi utilizado a abordagem do provedor de nuvem carregar o módulo de criptografia para o cliente, esse módulo está inserido dentro da aplicação que está executando no cliente, pois o provedor da nuvem não precisa conhecer como é feita a criptografia dos dados, já que o objetivo da criptografia é de manter os dados seguros contra o acesso indevido, seja o acesso do provedor ou de ataques ao ambiente da nuvem.

## 3.4 SECUREDROPBOX: A FILE ENCRYPTION SUITABLE FOR CLOUD STORAGE SERVICES

No trabalho de Chen et al. (2013) foi realizado o desenvolvimento de um sistema capaz de cifrar as informações dos usuários para evitar a violação de dados confidencias de usuários de determinados serviços, devido à falta de confidencialidade dos serviços de armazenamento em nuvem.

O sistema denominado SecureDropbox criptografa as informações do usuário de

forma automática, utilizando o algoritmo AES com chave de 128 *bits*, e mantém os arquivos criptografados sincronizados com a nuvem. Para realizar essa tarefa o SecureDropbox é composto por quatro módulos, que são o módulo de geração de chaves, módulo de gestão de chaves, módulo de sincronização e um módulo de criptografia de arquivos (CHEN et al., 2013).

Para utilizar o SecureDropbox, o usuário precisa informar um *e-mail* e uma senha personalizada. Então o sistema utiliza essas informações, juntamente com a chave secreta do sistema para gerar o que é chamado de SKUser e em seguida o sistema transforma o SKUser em um texto com formato de sequência de caracteres ASCII[18] com um esquema de codificação Base64[19]. Após a geração do SKUser, o mesmo é enviado para a conta de *e-mail* que o usuário do SecureDropbox atribuiu (CHEN et al., 2013).

A sincronização dos arquivos é realizada através de dois diretórios na máquina do usuário, no qual um diretório contém os arquivos em texto puro e o outro mantém os arquivos cifrados, que por sua vez são os arquivos sincronizados com a nuvem de armazenamento. Para evitar problemas entre os arquivos, é adicionado a extensão *.fes* nos arquivos armazenados no diretório que contém os arquivos cifrados (CHEN et al., 2013).

No presente trabalho, foram implementadas duas uma ferramentas que possuem o mesmo objetivo do trabalho realizado por Chen et al. (2013) de manter a confidencialidade dos arquivos armazenados em nuvens de armazenamento. Porém, existem algumas diferenças, como o armazenamento das chaves, no qual o SecureDropbox envia à chave através de um *e-mail* e no presente trabalho foi desenvolvido uma aplicação móvel denominada KeyManager para armazenar e gerenciar as chaves criptográficas. Outra diferença é que os arquivos criptografados do SecureDropbox são mantidos na máquina do cliente, enquanto a ferramenta desenvolvida utiliza uma base de dados que contém informações necessárias para a sincronização dos arquivos, e por isso, não há necessidade de manter os arquivos criptografados na máquina do usuário da ferramenta.

18 Código Padrão Norte-americado para Intercâmbio de Informações.
19 Método para codificação de dados para transferência na Internet.

Por fim, os trabalhos relacionados apresentam problemas de confidencialidade sobre as informações armazenadas em nuvens de armazenamento e sugerem a utilização da abordagem de criptografar as informações na máquina do cliente antes de serem enviadas à nuvem de armazenamento. Também foi encontrada uma ferramenta que utiliza essa abordagem, que é a SecureDropbox. Dessa forma, o trabalho realizado utilizou essa abordagem para implementar duas ferramentas, no qual foi implementado uma ferramenta denominada AntiNSA, que sincroniza as informações com a nuvem de armazenamento e diferentemente da ferramenta SecureDropbox que utiliza o e-mail para armazenar as chaves criptograficas, foi implementado um aplicativo para dispositivos móveis que pode ser utilizado para armazenar as chaves criptográficas utilizadas pela ferramenta AntiNSA.

# 4 DESENVOLVIMENTO

Devido aos problemas relacionados à falta de confidencialidade em sistemas de armazenamento em nuvem, citados anteriormente, é possível compreender a necessidade de sistemas capazes de manter dados criptografados. Para melhorar à questão de confidencialidade em sistemas de armazenamento em nuvem, foram desenvolvidas duas ferramentas que são a AntiNSA, responsável por sincronizar os arquivos com a nuvem, e o aplicativo móvel KeyManager que realiza o gerenciamento das chaves criptográficas utilizadas pela ferramenta AntiNSA.

Neste capítulo são abordados a arquitetura do trabalho e o funcionamento das ferramentas desenvolvidas, que tem por objetivo fornecer confidencialidade para os arquivos dos usuários que utilizam serviços de armazenamento em nuvem.

## 4.1 ARQUITETURA

Para minimizar a falta de confidencialidade em ambiente de armazenamento em nuvem, o presente trabalho propõe uma arquitetura similar a arquitetura utilizada por Chen et al. (2013). A arquitetura proposta no presente trabalho permite que as operações de criptografia e gerenciamento de chaves criptográficas sejam realizadas sem a necessidade de qualquer interação com a nuvem de armazenamento. Com essa arquitetura, a nuvem de armazenamento cumprirá o seu papel, que é somente armazenar as informações do usuário, sem ter acesso ao conhecimento do conteúdo das informações, pois o provedor da nuvem terá acesso somente às informações criptografadas do usuário.

A arquitetura proposta é apresentada na Figura 6, e consiste em duas partes: AntiNSA e KeyManager. No qual a AntiNSA (Desktop) tem por objetivo manter as informações descriptografadas na máquina do usuário sempre sincronizadas com a nuvem de armazenamento, já a KeyManager (Mobile) gerencia as chaves criptográficas utilizadas pelo usuário através da AntiNSA.

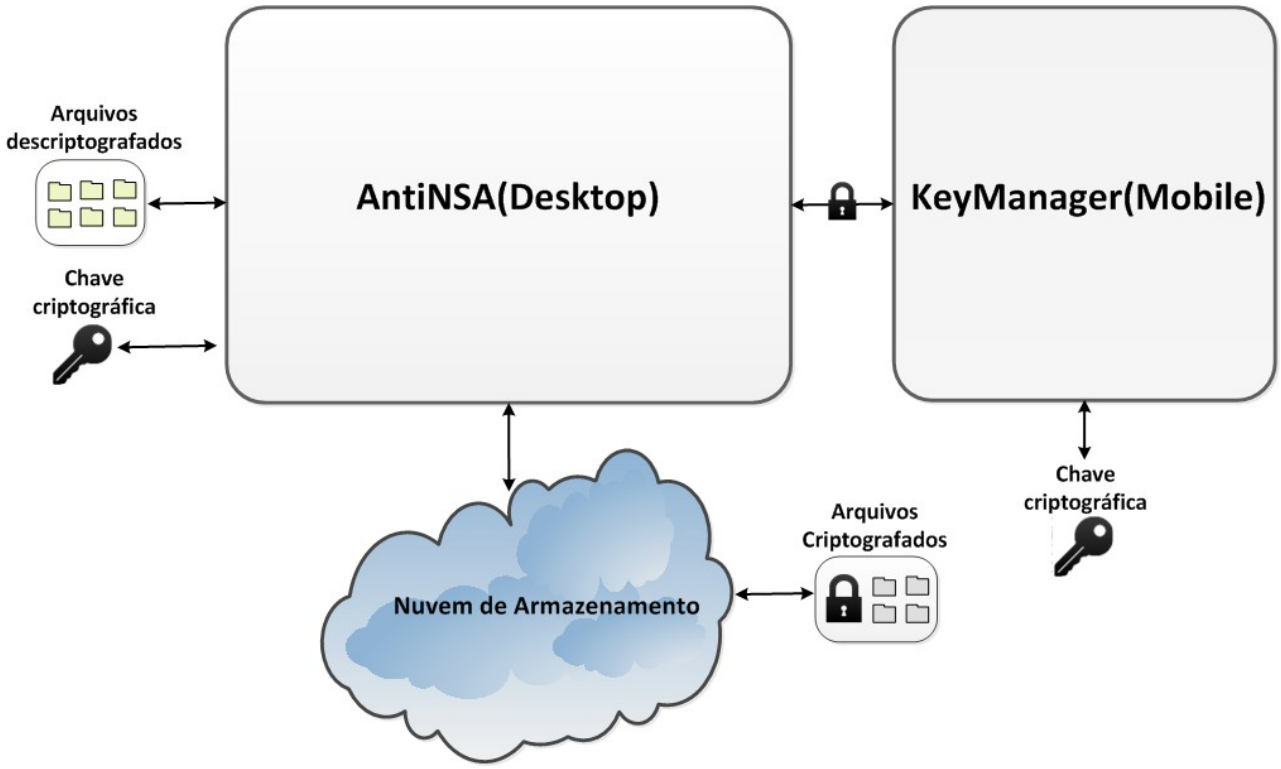


Figura 6 – Arquitetura para melhorar a confidencialidade em nuvens de armazenamento.

O serviço de armazenamento em nuvem, é utilizado pela AntiNSA para armazenar as informações do utilizador e torná-las altamente disponíveis, no qual essas informações são mantidas na nuvem de armazenamento de forma criptografada.

A interação entre a AntiNSA e a KeyManager é necessária para que possa ser realizado o envio e o recebimento da chave criptográfica, no qual esse procedimento ocorre através de um canal de comunicação seguro que utiliza o protocolo *Secure Sockets Layer* (SSL), de forma que seja mantida a a integridade e confidencialidade da chave.

Por fim, com essa arquitetura a ferramenta prove melhorias em relação a falta de confidencialidade no armazenamento de arquivos em serviços de armazenamento em nuvem, pelo fato das informações estarem armazenadas de uma forma ilegível para o provedor do serviço de armazenamento em nuvem.

## 4.2 A FERRAMENTA

Para realizar a implementação da arquitetura foram desenvolvidos dois protótipos, a ferramenta AntiNSA que corresponde a parte AntiNSA da arquitetura e o aplicativo móvel KeyManager que corresponde à parte KeyManager. Também foi necessário realizar a seleção de um serviço de armazenamento em nuvem, no qual foi selecionado o Google Drive, por fornecer uma API de acesso ao serviço de armazenamento e ser um dos serviços de armazenamento em nuvem mais conhecidos.

O código fonte das ferramentas AntiNSA e KeyManager podem ser encontrados nos respectivos Apêndices A e B e também estão disponíveis em https://github.com/juniorcesar/AntiNSA .

As Figuras 7 e 8 apresentam respectivamente as principais telas da ferramenta AntiNSA e do aplicativo móvel KeyManager.

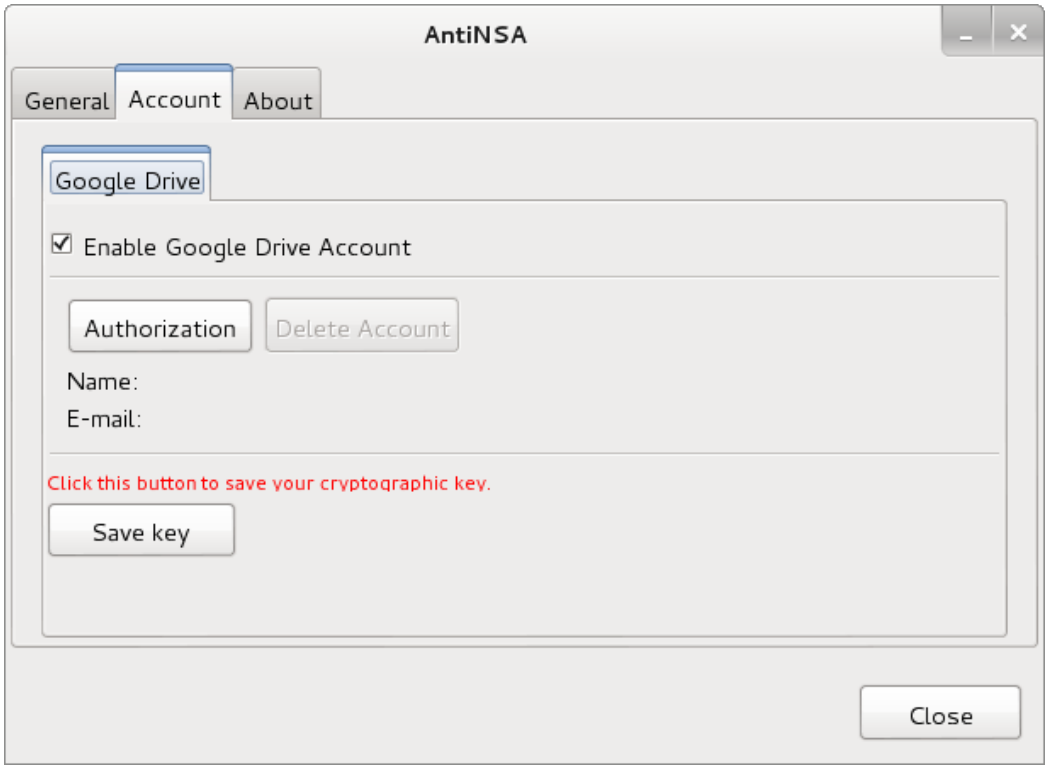


Figura 7 – Tela de configuração de conta da ferramenta AntiNSA.

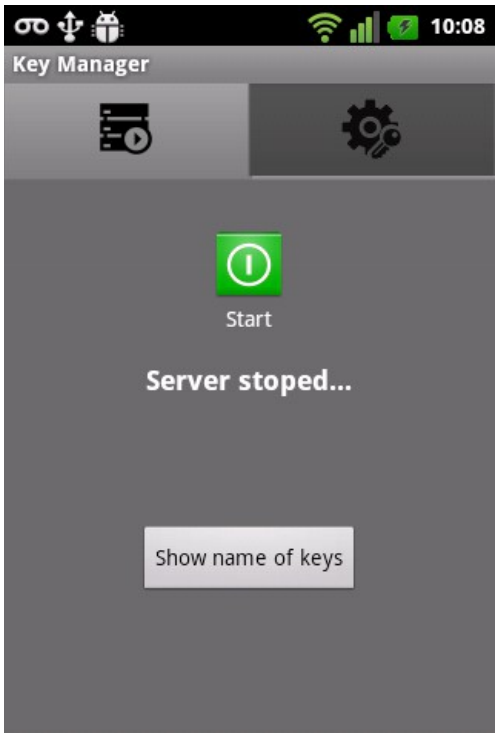

Figura 8 – Tela principal do aplicativo móvel KeyManager.

A Figura 9 apresenta os detalhes das ferramentas desenvolvidas, no qual a ferramenta AntiNSA é subdividida em três módulos que são:

- Módulo de autorização: responsável por obter a autorização de acesso a nuvem de armazenamento;
- Módulo de segurança: realiza as operações de criptografia nos arquivos que são sincronizados com a nuvem;
- Módulo de sincronização: mantém os arquivos do usuário sincronizados com a nuvem de armazenamento;

Além dos módulos, a AntiNSA contém os seguintes elementos:

- Configuração: utilizado para armazenar a chave criptográfica, arquivos de configuração e os arquivos utilizados para obter acesso ao serviço de armazenamento;
- Arquivos: utilizado para armazenar os arquivos que o usuário deseja manter sincronizado com a nuvem de armazenamento;
- Cache: utilizado para armazenar os arquivos temporários;
- Base de dados: utilizado para armazenar informações dos arquivos sincronizados;
- Cliente do servidor de chaves: responsável por enviar a chave criptográfica gerada

pela aplicação e receber a chave da KeyManager quando for necessário.

E o aplicativo móvel KeyManager é composto pelos seguintes elementos:

- Servidor de chaves: responsável por receber e enviar chaves criptográficas para a AntiNSA;
- Gerenciamento de chaves: responsável por armazenar as chaves recebidas através do servidor de chaves e fornecê-las quando necessário;
- Armazenamento de chaves: responsável pelo armazenamento das chaves.

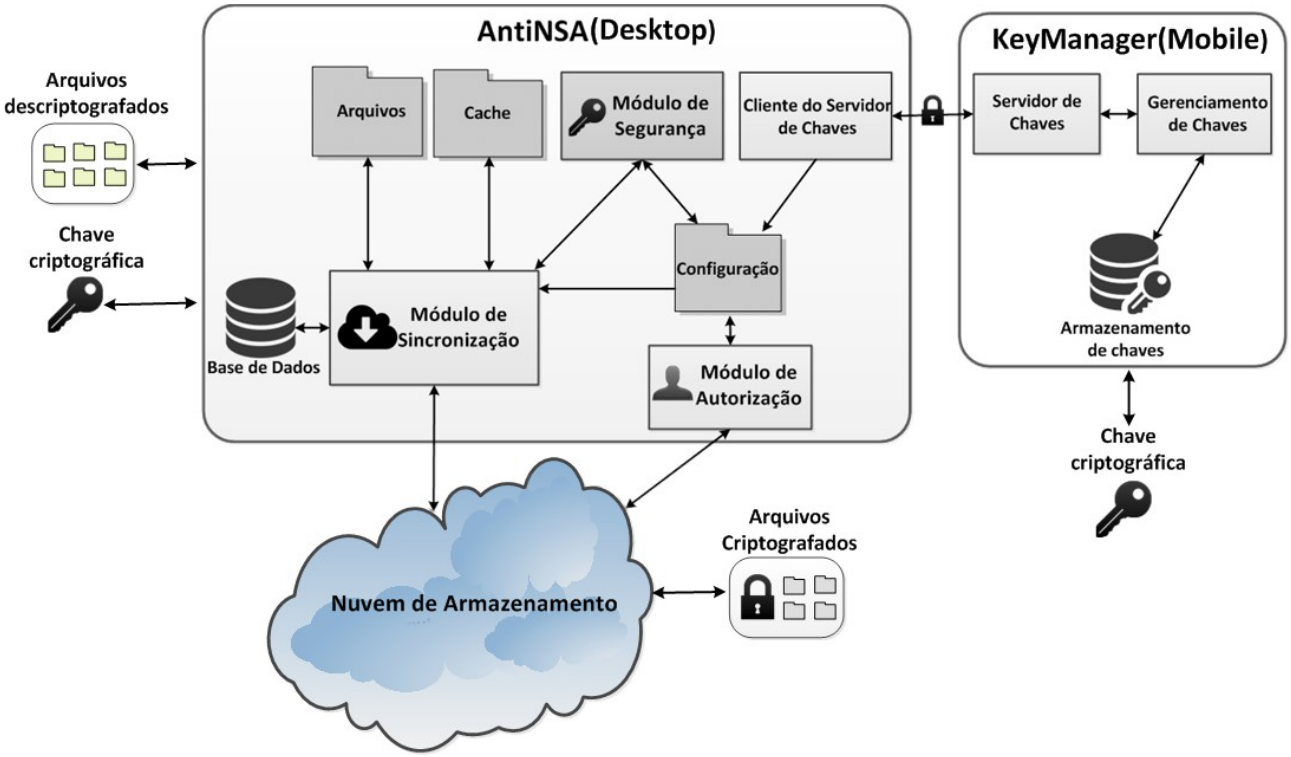


Figura 9 – Detalhes das ferramentas AntinNSA e KeyManager.

A ferramenta AntiNSA utiliza os recursos do sistema de armazenamento em nuvem do Google Drive, que disponibiliza uma API de comunicação para que a aplicação possa utilizar os recursos do armazenamento em nuvem. Dessa forma, para que o usuário da ferramenta AntiNSA possa fazer o uso da aplicação é necessário possuir uma conta Google[20] e realizar o processo de autorização, que fornece permissão para que a ferramenta AntiNSA possa se comunicar com a nuvem de armazenamento.

Além de realizar a autorização de acesso a nuvem, é necessário gerar a chave criptográfica e enviá-la para o aplicativo KeyManager, pois não é suficiente manter a

20https://accounts.google.com/SignUp

chave armazenada somente na máquina em que a aplicação está utilizando, devido ao fato de que a chave é gerenciada pelo usuário, e caso o usuário perca a chave não será possível recuperá-la. Assim, o aplicativo KeyManager visa auxiliar na questão do gerenciamento da chave, armazenando-a de forma segura e para que a mesma possa ser disponibilizada para o usuário de acordo com a necessidade do mesmo.

O processo de sincronização é iniciado, somente após a execução do processo de autorização e geração de chave criptográfica. Dessa forma, o módulo de sincronização sincroniza os arquivos que são armazenados no diretório padrão da aplicação, chamado de AntiNSA. Além do diretório AntiNSA, a ferramenta mantém um diretório oculto com de `.cache`, que armazena os arquivos criptografados pelo módulo de segurança, no momento de envio de um arquivo e quando um arquivo criptografado é descarregado da nuvem para ser descriptografado no diretório AntiNSA. Porém, os arquivos criptografados do diretório `.cache` são removidos logo após sua utilização, devido ao fato de que eles não serão mais úteis para o usuário. Isso ocorre porque a ferramenta AntiNSA utiliza uma base dados, que armazena todas informações necessárias dos arquivos criptografados em uma tabela, para que posteriormente, possam ser utilizadas nas comparações realizadas pelo módulo de sincronização, logo, não é necessário manter os arquivos criptografados na máquina do usuário, devido ao fato dos arquivos criptografados serem ilegíveis para o usuário, dessa forma, esses arquivos estariam ocupando espaço no disco rígido da máquina do usuário.

Nas próximas seções, são abordadas em detalhes as tecnologias que foram utilizadas para realizar implementação da arquitetura proposta, a implementação do módulo de autorização, módulo de segurança, o módulo de sincronização e a forma que é gerenciada a chave criptográfica utilizada pela aplicação.

### 4.2.1 Tecnologias utilizadas

No desenvolvimento deste trabalho foram utilizadas várias tecnologias, devido ao fato de envolver duas aplicações, sendo uma aplicação *desktop* e outra para dispositivo móvel. Ambas foram desenvolvidas utilizando a linguagem de programação Java.

A ferramenta AntiNSA foi desenvolvida utilizando o ambiente integrado de desenvolvimento NetBeans. A interface gráfica da aplicação AntiNSA foi desenvolvida em swing e awt[21]. O banco de dados utilizado para armazenamento das informações dos arquivos foi o SQLite[22], no qual o mesmo foi utilizado por não haver necessidade de instalar um servidor de banco de dados separado da aplicação. Para realizar a integração com a nuvem de armazenamento do Google foram utilizadas as APIs OAuth 2.0[23] para realizar a autenticação no serviço de armazenamento e a Google Drive API[24] para gerenciar os arquivos armazenados na nuvem.

O aplicativo móvel KeyManager foi desenvolvido no ambiente integrado de desenvolvimento Eclipse. O Eclipse fornece um ambiente para desenvolvimento de aplicativos para a plataforma Android, que facilita o desenvolvimento de aplicações móveis. Para desenvolver o aplicativo KeyManager foi utilizado a API Android 4.3[25].

### 4.2.2 Módulo de autorização

A ferramenta AntiNSA realiza a comunicação com a nuvem de armazenamento somente se estiver autorizada a gerenciar arquivos de um determinado usuário do serviço de armazenamento, no caso o Google Drive.

A API do Google Drive fornece uma maneira segura e eficiente para realizar a autorização de acesso ao serviços do Google, através do protocolo de autorização OAuth. Para isso, foi necessário registrar a ferramenta AntiNSA no Google Developers[26] e gerar o ID do cliente, utilizado pela ferramenta ao solicitar o acesso ao serviço de armazenamento em nuvem. Dessa forma módulo de autorização utiliza os dados fornecidos pelo Google Developers, que são o ID (identificação) do cliente e o cliente secreto (>> o que é cliente secreto?) da ferramenta, para solicitar as credenciais de acesso ao serviço de armazenamento. Assim, cada usuário que utilizar a aplicação deverá realizar a

21 http://docs.oracle.com/javase/7/docs/technotes/guides
22 http://www.sqlite.org/
23 https://developers.google.com/accounts/docs/OAuth2
24 https://developers.google.com/drive/v2/reference/
25 https://developer.android.com/about/versions/android-4.3.html
26 https://developers.google.com

autorização para que a ferramenta AntiNSA possa gerenciar os arquivos armazenados no Google Drive.

O processo de autorização consiste na configuração de uma conta para que a ferramenta AntiNSA obtenha acesso ao serviço de armazenamento em nuvem. No momento em que o usuário estiver realizando a configuração de uma conta na ferramenta AntiNSA, é aberto um navegador que solicita o acesso à uma conta Google. Após o usuário conectar sua conta, é aberta uma página de autorização de uso do Google Drive pela ferramenta AntiNSA. Assim, o usuário deve realizar a autorização, para que a ferramenta receba e armazene as credenciais de acesso no diretório de configuração da aplicação, pois as mesmas são necessárias para obter acesso ao serviço de armazenamento e gerenciar os arquivos que serão armazenados futuramente.

Por fim, o módulo de autorização permite que o módulo de sincronização da ferramenta AntiNSA acesse o serviço de armazenamento em nuvem utilizando as credenciais de acesso que foram recebidas no momento da autorização realizada pelo usuário.

### 4.2.3 Gerenciamento de chaves criptográficas

Para melhorar a questão da disponibilidade foi desenvolvido um aplicativo móvel, denominado KeyManager, capaz de armazenar, enviar e receber as chaves criptográficas utilizadas pela ferramenta AntiNSA de uma forma segura. O processo de transferência das chaves criptográficas são realizados através de canais de comunicação seguros, no qual foi utilizado o protocolo SSL, através da classe `javax.net.ssl.SSLServerSocket`.

A chave criptográfica gerada pela ferramenta AntiNSA é única, e deve ser armazenada em outro local para melhorar a disponibilidade dos dados gerenciados pela ferramenta, em caso do usuário realizar a instalação da ferramenta em outra máquina, ou até mesmo em casos de desastres, no qual o usuário pode perder a chave que está armazenada juntamente com a instalação da ferramenta.

Na primeira utilização da aplicação, após o usuário ter realizado a autorização do

uso da aplicação para gerenciar arquivos no Google Drive, é gerado a chave criptográfica que será utilizada pelo módulo de segurança para realizar as devidas operações nos arquivos, no qual a chave é armazenada no diretório de configuração da aplicação. Em seguida é recomendado que o usuário armazene a chave em outro local na própria máquina ou que envie a chave para o aplicativo KeyManager, que é a forma mais indicada para armazenar a chave, devido ao fato de que a chave estará sob controle do usuário e disponível para utilizações futuras. A Figura 10 mostra o procedimento necessário para que o usuário envie a chave gerada pela ferramenta AntiNSA para o aplicativo KeyManager.

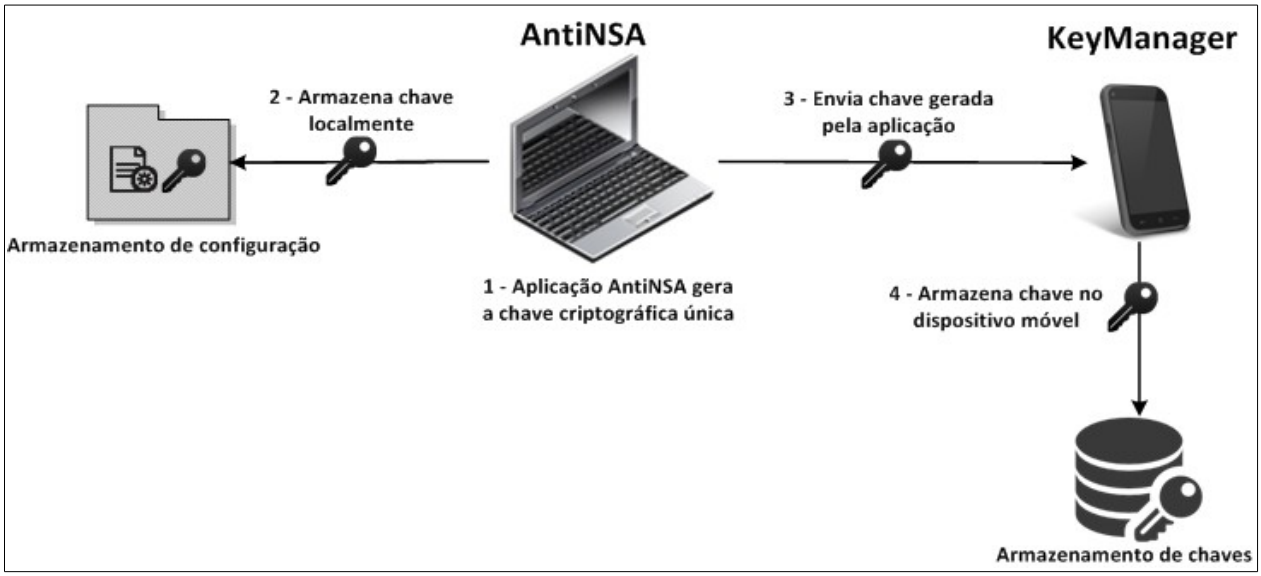


Figura 10 – Geração de chave criptográfica.

O processo de transferência deve ser inicializado pelo próprio usuário, no qual o mesmo deve possuir um celular com sistema operacional Android e possuir o aplicativo KeyManager instalado. Dessa forma, o usuário deverá iniciar o servidor no celular, no qual será informado o endereço IP em que o servidor estará disponível. Em seguida, o usuário deve informar o endereço IP do servidor na aplicação AntiNSA e informar um nome para a chave que será enviada, e no momento em que a chave é enviada, é necessário que o armazenamento da chave seja aprovado pelo usuário no aplicativo KeyManager, e após a autorização do usuário, a chave é armazenada no armazenamento interno do celular.

A Figura 11 exibe o processo de recuperação da chave criptográfica gerenciada pelo aplicativo KeyManager. Para realizar essa recuperação, é necessário que o usuário inicialize o servidor do aplicativo KeyManager, em seguida deve ser informado o endereço do servidor e o nome da chave desejada, para que a aplicação AntiNSA possa realizar a

solicitação da chave. Após a solicitação o usuário receberá uma mensagem no aplicativo KeyManager, informando que a chave está sendo solicitada e se o usuário permite que a chave seja enviada para o solicitante. Caso o usuário autorize o envio, a chave é recuperada do armazenamento de chaves do celular e enviada para a aplicação AntiNSA, que armazena a chave no diretório de configuração da aplicação. Por fim, com a chave recuperada, o processo de sincronização está pronto para ser iniciado e realizar as operações de criptografia e descriptografia dos arquivos gerenciados pela aplicação.


Figura 11 – Recuperação da chave criptográfica.

Por fim, o armazenamento da chave criptográfica realizado através do aplicativo KeyManager permite que o usuário possa realizar outras instalações da ferramenta AntiNSA para acessar seus dados armazenados na nuvem a qualquer momento, desde que tenha acesso a Internet. Dessa forma a disponibilidade das informações do usuário será mantida, devido o fato do usuário ter acesso às suas informações de qualquer local.

### 4.2.4 Módulo de segurança

O módulo de segurança é o responsável por manter a confidencialidade dos arquivos armazenados na nuvem, que é realizado através de técnicas de criptografia. No presente trabalho foi utilizado o algoritmo de criptografia AES com uma chave de 128 *bits.* O AES foi escolhido por ser um padrão em algoritmos de criptografia.

A implementação do algoritmo AES no módulo de segurança da aplicação não foi

necessário utilizar nenhuma outra Interface de Programação de Aplicativos (API), devido ao fato da implementação padrão do Java possuir alguns recursos de segurança disponíveis para o uso.

A Figura 12 exibe as linhas de código responsáveis pela geração da chave única, utilizada por todo processo de criptografia e descriptografia realizado pela aplicação. A classe `KeyGenerator`[27] possui os métodos que realizam a geração de chaves, e para isso é necessário obter uma instância da implementação de um algoritmo, no caso foi utilizado o *AES*, em seguida é informado o comprimento da chave criptográfica, que é de 128 *bits*. Por fim, a classe `KeyGenerator` realiza a chamada do método `generateKey()` que tem por função gerar a chave criptográfica e armazená-la em um objeto do tipo `SecretKey`[28], em seguida o objeto `SecretKey` é armazenado em um arquivo específico de armazenamento de chaves criptográficas, como está sendo mostrado na Seção 5.3.

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
keygen.init(128);
SecretKey key = keygen.generateKey();
```

Figura 12 – Geração da chave criptográfica.

O processo de criptografia é realizado por uma classe chamada `SecretKeyAESCrypto`, que possui um método construtor, um método chamado `encrypt` e outro `decrypt`.

A Figura 13 exibe o método construtor da classe `SecretKeyAESCrypto` que realiza o carregamento da chave `SecretKey` utilizada pelos métodos `encrypt` e `decrypt` para realização de suas respectivas operações com os arquivos. O construtor também obtém uma instância do algoritmo AES, utilizada pela classe `Cipher` que é responsável por inserir a chave e o tipo do processo, se um arquivo será criptografado ou descriptografado.

```
public class SecretKeyAESCrypto {

    private Cipher cipher;
    private SecretKey key;

    public SecretKeyAESCrypto() throws Exception {
        key = KeyManager.loadKey();
        cipher = Cipher.getInstance("AES");
    }
```

Figura 13 – Classe `SecretKeyAESCrypo.`

27 http://docs.oracle.com/javase/7/docs/api/javax/crypto/KeyGenerator.html
28 http://docs.oracle.com/javase/7/docs/api/javax/crypto/SecretKey.html

O processo de criptografia dos arquivos é realizado pelo método `encrypt`, exibido na Figura 14. O método `encrypt`, tem como entrada um objeto `File`, no caso o arquivo que será criptografado e tem como saída outro objeto `File` com o conteúdo do objeto de entrada, porém criptografado, no qual o arquivo criptografado será removido do diretório local após a aplicação ter armazenado os dados do arquivo no banco de dados e ter enviado o mesmo para a nuvem de armazenamento.

```
public File encrypt(File file) {
    try {
        cipher.init(Cipher.ENCRYPT_MODE, key);

        BufferedInputStream bufferedInputStream = new BufferedInputStream(new FileInputStream(file));
        CipherInputStream cipherInputStream = new CipherInputStream(bufferedInputStream, cipher);
        File tempFile = new File(GDUtils.CACHE_DIR + "/" + file.getName());

        BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(new FileOutputStream(tempFile));

        int read = 0;
        byte[] bytes = new byte[1024];
        while ((read = cipherInputStream.read(bytes)) != -1) {
            bufferedOutputStream.write(bytes, 0, read);
        }

        bufferedOutputStream.close();
        bufferedInputStream.close();
        cipherInputStream.close();

        return tempFile;
```

Figura 14 – Método responsável pelo processo de criptografia dos arquivos.

O processo de criptografia realizado pelo método `encrypt` utiliza a classe `Cipher` para informar qual operação será realizada pelo processo, isso ocorre na chamada do método `init(Cipher.ENCRYPT_MODE, key)`, no qual o primeiro parâmetro é a constante `Cipher.ENCRYPT_MODE` informa que a operação que será realizada é de criptografia e o segundo parâmetro `key` é a chave que será utilizada por esse processo. Após ter a classe `Cipher` inciada com o modo de criptografia, é utilizado a classe `BufferedInputStream` para realizar a leitura do arquivo a ser criptografada. O arquivo armazenado no objeto `BufferedInputStream` e o objeto `Cipher` são utilizados como entrada no método construtor da classe `CipherInputStream`, que é a classe responsável por realizar a operação de criptografia a partir dos dados que foram informados no momento da inicialização de sua instância. Para que o objeto `CipherInputStream` possa criptografar o arquivo é necessário criar outro arquivo que receberá esse conteúdo criptografado, para isso é criado um arquivo temporário e armazenado em um diretório local da máquina, no qual esse arquivo é carregado em um objeto `BufferedOutputStream` que realizará a escrita dos *bytes* criptografados no

arquivo temporário. No momento em que o objeto da classe `CipherInputStrem` realiza a chamada do método `read(bytes)`, é retornado uma sequência de *bytes* criptografados, correspondente ao conteúdo original do arquivo armazenado no objeto `BufferInputStream`, e esses *bytes* são escritos no arquivo temporário. No fim da operação de criptografia do arquivo, os fluxos de entrada e saída são fechados e o método `encrypt` retorna o arquivo temporário, que possui agora o conteúdo do arquivo recebido como entrada no método, porém criptografado.

A Figura 15 exibe o processo de descriptografia de arquivos, realizado pelo método denominado `decrypt`. Esse método recebe como entrada um objeto `File` que contém um arquivo criptografado e retorna um arquivo descriptografado.

```
public File decrypt(File file) {
    try {

        cipher.init(Cipher.DECRYPT_MODE, key);
        BufferedInputStream bufferedInputStream = new BufferedInputStream(new FileInputStream(file));
        CipherInputStream cipherInputStream = new CipherInputStream(bufferedInputStream, cipher);

        File decryptedFile = new File(Config.STORE_DEFAULT.getAbsolutePath() + "/" + file.getName());
        BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(new FileOutputStream(decryptedFile));

        int read = 0;
        byte[] bytes = new byte[1024];
        while ((read = cipherInputStream.read(bytes)) != -1) {
            bufferedOutputStream.write(bytes, 0, read);
        }

        bufferedOutputStream.close();
        bufferedInputStream.close();
        cipherInputStream.close();

        return decryptedFile;
```

Figura 15 – Método responsável pelo processo de descriptografia dos arquivos

Para descriptografar um determinado arquivo a classe `Cipher` realiza a chamada do método `init(Cipher.DECRYPT_MODE, key)` e informa como parâmetro o modo de descriptografia através da constante `Cipher.DECRYPT_MODE` e a mesma chave utilizada no processo de criptografia do arquivo. Em seguida é utilizado o mesmo procedimento utilizado pelo método `encrypt`, com a diferença é que o arquivo que será retornado, já será armazenado no diretório de armazenamento padrão da aplicação AntiNSA, que contém os arquivos descriptografados gerenciados pela aplicação. Por fim, é realizado o procedimento descriptografia e escrita no arquivo de saída, que terá o conteúdo do arquivo criptografado, porém esse conteúdo estará totalmente legível para o usuário da aplicação.

### 4.2.4 Módulo de sincronização

O módulo de sincronização tem por função manter os arquivos gerenciados pela aplicação AntiNSA sempre atualizados. Esse módulo é executado após a realização do processo de autorização, assim a sincronização permanece ativa enquanto a ferramenta AntiNSA estiver em execução na máquina do usuário. Nesta seção é apresentado o fluxo necessário para que o módulo de sincronização possa realizar essa tarefa.

O processo de sincronização é executado em uma única `Thread`[29], no qual a aplicação inicia o processo de sincronização verificando se há conexão com a Internet, caso não possua, a `Thread` não executa sincronização. Caso haja conexão, são executados os dois principais processos da aplicação: o processo de comparação dos arquivos da nuvem com a base de dados e o processo de comparação dos arquivos locais com a base de dados, mostrados em forma de fluxograma nas Figuras 11 e 16, respectivamente.

A Figura 16 exibe o primeiro processo executado pela aplicação para realizar a sincronização dos arquivos. Esse processo se inicia com a leitura dos arquivos que estão armazenados na nuvem, isso se faz através da API do Google Drive[30], fazendo o uso da classe `Drive` que fornece uma lista de objetos `File` que contém todas as informações dos arquivos armazenados na nuvem. Além disso a classe `Drive` é responsável por realizar as operações básicas como criação, remoção, alteração e listagem de arquivos. A segunda lista utilizada no processo de comparação é resultado de uma consulta da tabela do banco de dados, que contém as informações dos arquivos armazenados no diretório local da máquina do usuário, essas informações são armazenadas em uma lista de objetos da própria aplicação chamado `DataFile`, que contém a mesma estrutura da tabela do banco de dados `data_file`, no qual, todas as informações dos arquivos sincronizados são mantidas nessa tabela.

29 http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html
30 https://developers.google.com/drive/v2/reference/

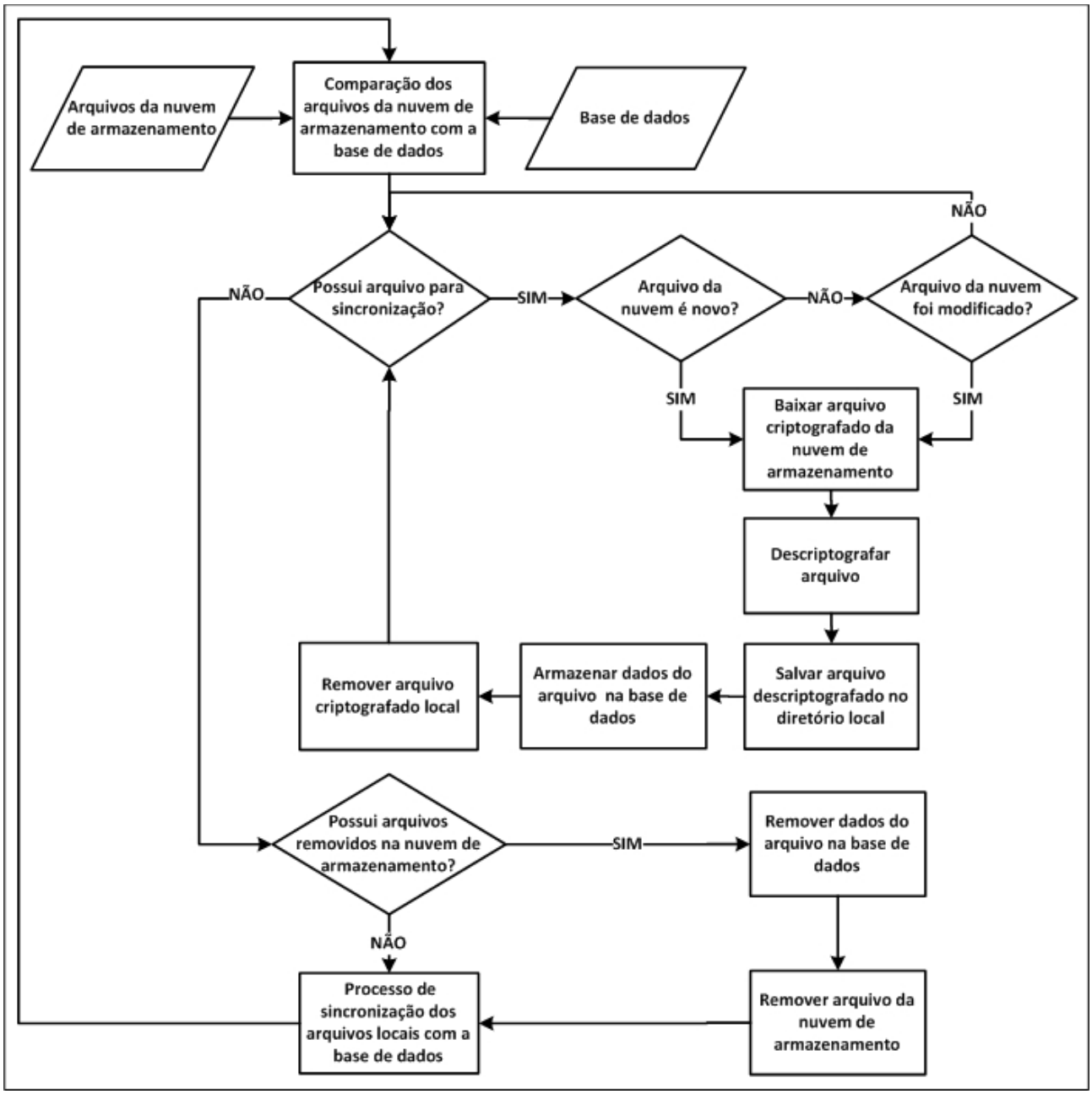

Figura 16 – Processo de sincronização da nuvem com a base de dados.

A Figura 17 exibe a tabela `data_file`, que armazena os seguintes dados: o `id`, que é um número de identificação único para o registro; `name`, que armazena o nome do arquivo; `date`, que armazena a data da última modificação do arquivo; `local_hash`, que armazena o *hash*[31] MD5 (*Message-Digest algorithm* 5) do arquivo local descriptografado; e o `cloud_hash`, utilizado para armazenar o código *hash* MD5 que a API do Google Drive fornece através do método `getMd5Checksum()` do objeto `File` API.

31 Uma função *hash* é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo.

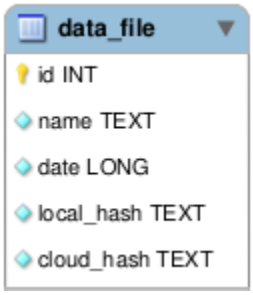

Figura 17 - Tabela `data_file` do banco de dados.

Após obter a lista dos arquivos armazenados na nuvem e a listagem dos registros da tabele `data_file`, a aplicação inicia o processo de comparação. Ao utilizar a aplicação AntiNSA pela primeira vez, não haverá nenhum arquivo no diretório AntiNSA criado pela aplicação na nuvem de armazenamento, porém, caso o usuário já tenha utilizado a aplicação e deseja instalar a mesma em outro computador, poderá existir arquivos na nuvem que deverão ser sincronizados para esse novo dispositivo. Por esse motivo, o processo de comparação dos arquivos da nuvem com a base de dados é executado por primeiro.

O primeiro passo desse processo é verificar se existem arquivos armazenados na nuvem que não estão armazenados na base de dados, caso isso ocorra, a aplicação entenderá que esses arquivos são novos, ou seja, ainda não estão armazenados localmente. Em seguida esses novos arquivos são obtidos e armazenados em um diretório temporário, devido ao fato de estarem criptografados são inutilizáveis pelo usuário, portanto não ficaram por muito tempo armazenados no diretório local. Dessa forma, o mesmo é descriptografado e armazenado no diretório local monitorado pela aplicação, em seguida a aplicação obtém as informações desse arquivo e armazena na base de dados. E como já foi dito, não há necessidade de manter o arquivo criptografado localmente, e após obter as informações necessárias para a comparações futuras o arquivo é removido do diretório local e o fluxo de comparação é retomado pela aplicação. A Figura 18 exibe o procedimento que ocorre quando um arquivo novo é encontrado na nuvem de armazenamento.

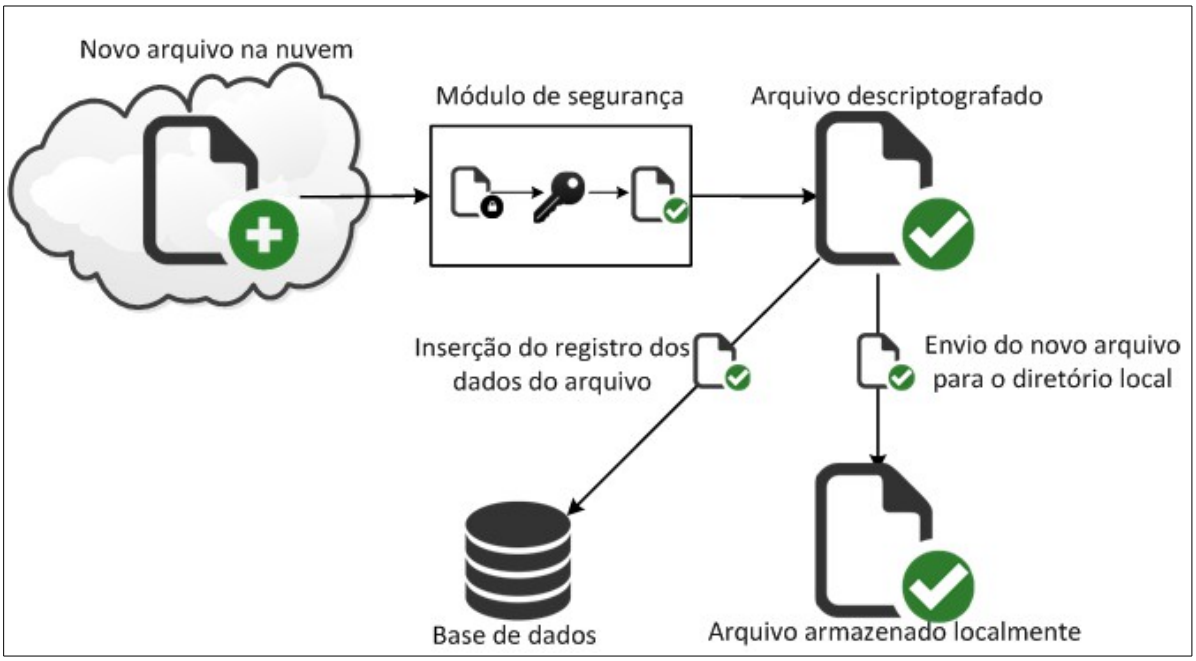

Figura 18 – Arquivo novo na nuvem de armazenamento.

No caso de um determinado arquivo já existir na base de dados é necessário comparar algumas informações do arquivo para verificar se há ou não modificações no mesmo e realizar a sincronização caso necessário. Para isso, são comparados a data de última modificação e o código *hash* do arquivo, dessa forma se ambas informações não forem iguais, o arquivo da nuvem é submetido ao mesmo processo que os arquivos novos são submetidos para serem armazenados localmente, como mostra a Figura 19. E caso não haja alteração no arquivo o fluxo de comparação é retomado pela aplicação.

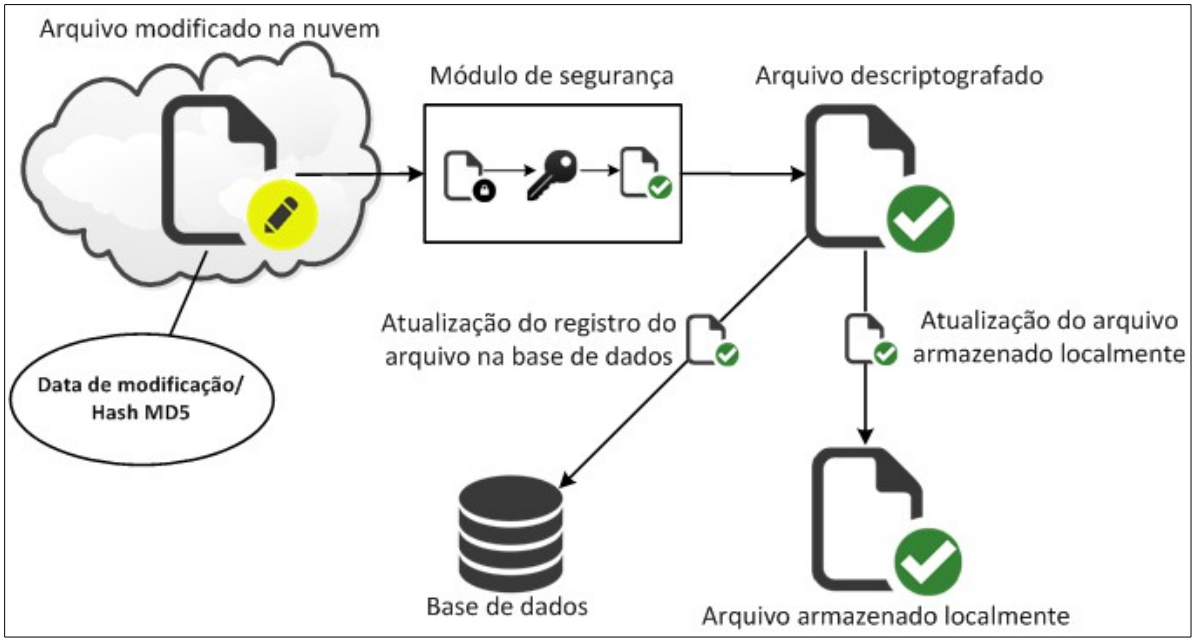

Figura 19 –Arquivo modificado na nuvem de armazenamento.

Quando não houver mais aquivos para serem comparados é iniciado um processo de verificação dos arquivos que foram removidos da nuvem de armazenamento e que ainda estão armazenados localmente. Se houverem arquivos removidos, o mesmo tem seu respectivo registro removido da base de dados e o arquivo é removido localmente, como mostra a Figura 20.

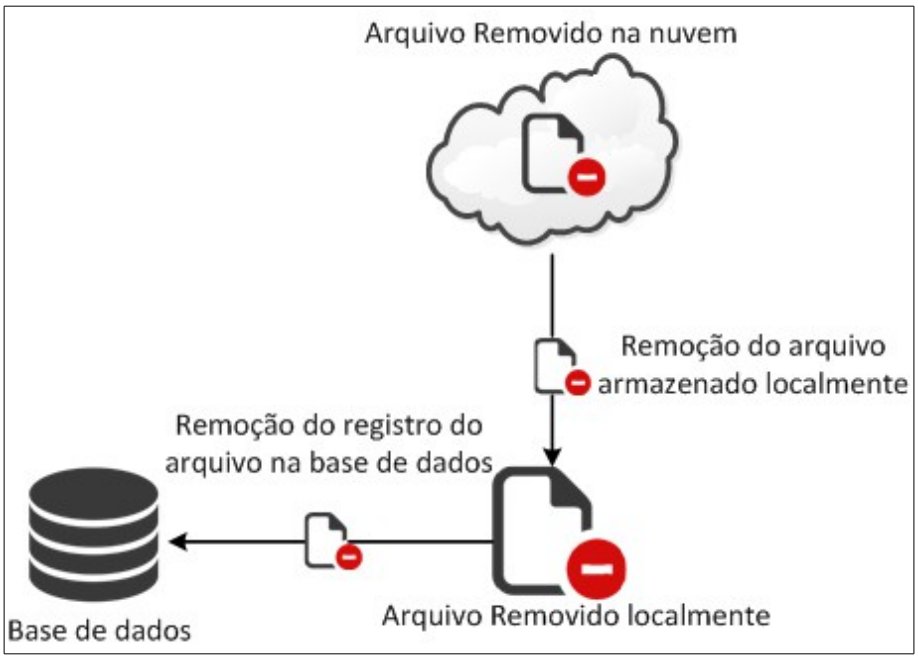


Figura 20 – Arquivo removido na nuvem de armazenamento.

Em seguida é iniciado o processo de comparação dos arquivos locais com a base de dados, como mostra a Figura 21. Esse processo se inicia logo após o processo de sincronização dos arquivos da nuvem com a base de dados, e recebe como entrada uma listagem dos arquivos locais armazenados no diretório padrão da aplicação, que é realizada através de um objeto `java.io.File`[32] que fornece o método `listFiles()`, no qual é retornado um vetor de objetos `java.io.File`. A outra entrada para comparação é uma listagem dos registros da tabela da base de dados `data_file` que são armazenadas em um objeto `DataFile`.

32 http://docs.oracle.com/javase/6/docs/api/java/io/File.html

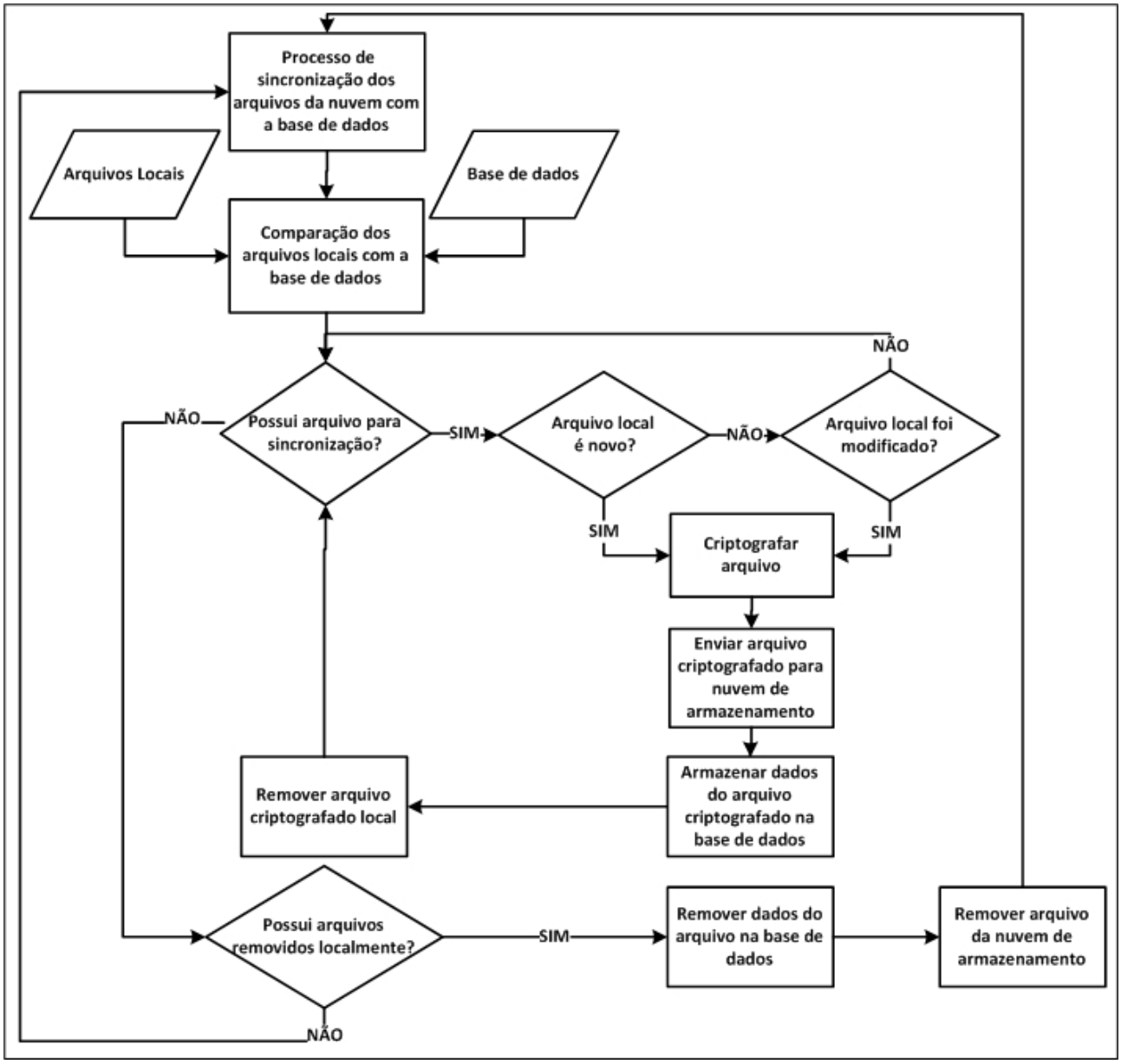

Figura 21 – Processo de sincronização dos arquivos locais com a base de dados.

Após obter as listagens, é verificado se existem arquivos para comparação, caso existam, é necessário verificar se esse arquivo é novo ou não, através de uma consulta no banco verificando sua existência. Se o arquivo em questão for novo, o mesmo é submetido ao processo de criptografia, que criptografa o arquivo e salva em um diretório temporário, para que posteriormente o arquivo criptografado seja enviado para a nuvem de armazenamento. Em seguida os dados do arquivo são armazenados na base de dados para comparações posteriores, o arquivo criptografado é removido localmente e é verificado novamente se existem arquivos para sincronização. A Figura 22 exibe o procedimento citado anteriormente.

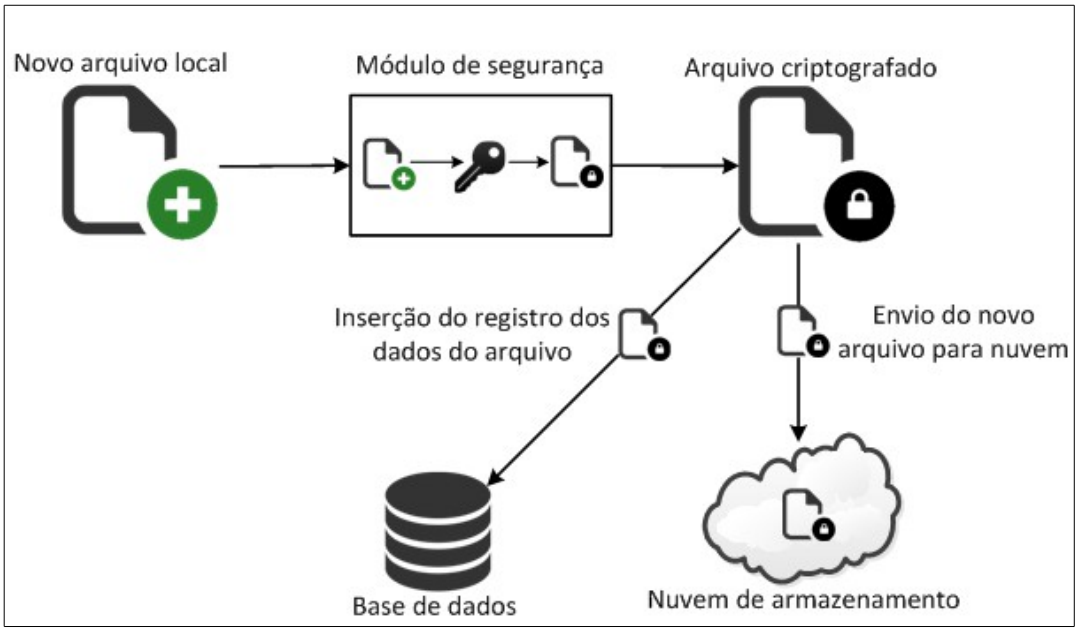

Figura 22 – Arquivo novo no diretório local.

Caso um determinado arquivo não seja novo, é verificado se o mesmo foi modificado, se não o fluxo da aplicação retorna para verificação de outros arquivos. Se o arquivo foi alterado, o mesmo será enviado para o módulo de segurança para ser criptografado, como está sendo mostrado na Figura 23, e como o arquivo alterado já existe na nuvem, ele não será enviado como um arquivo novo, pois a API do Google Drive fornece um método de atualização do arquivo existente, sendo necessário informar o número de identificação do arquivo fornecido pela classe `File` da API. Após o arquivo criptografado ter sido atualizado na nuvem de armazenamento, o mesmo tem seus dados atualizados na base e o arquivo criptografado é removido localmente.

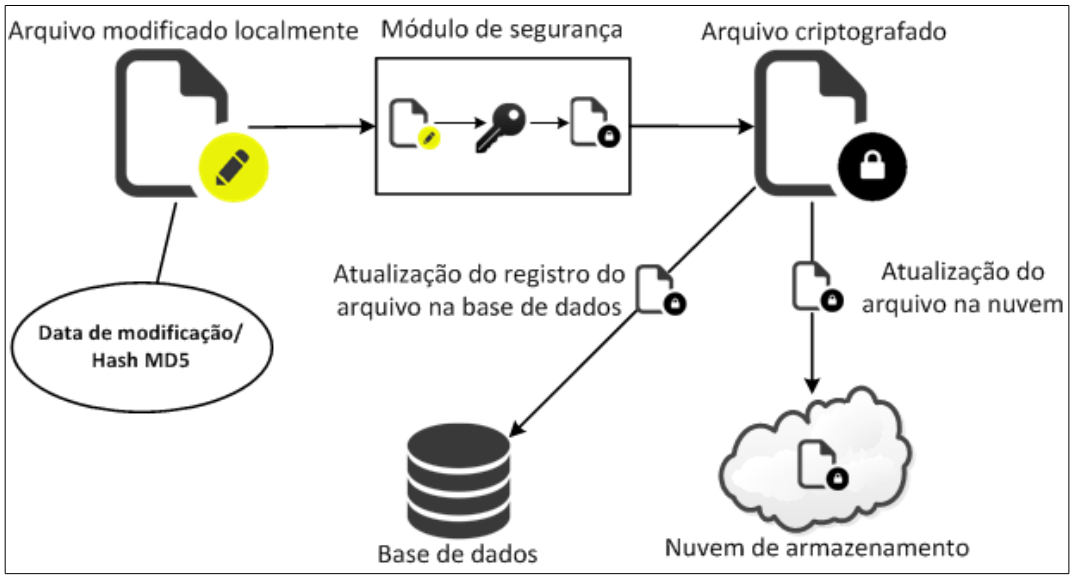

Figura 23 –Arquivo modificado no diretório local.

No momento em que não existirem mais arquivos para comparação, é verificado se existem arquivos que foram removidos localmente, mas ainda estão armazenados na base de dados e na nuvem de armazenamento. A Figura 24 exibe o procedimento de remoção. Caso existam arquivos que foram removidos localmente, o registro do arquivo é removido da base de dados e o arquivo criptografado é removido na nuvem de armazenamento, e em seguida o processo de comparação dos arquivos da nuvem com a base de dados é iniciado novamente, e o mesmo ocorre caso não existirem arquivos para remoção.

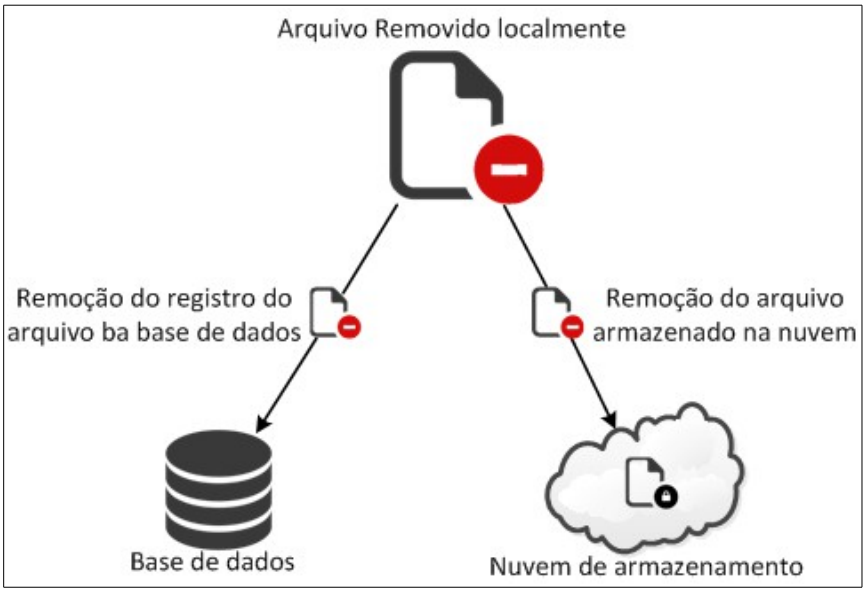


Figura 24 – Arquivo removido no diretório local.

O processo de sincronização será sempre executado enquanto a aplicação estiver em execução, exceto se a conta do usuário não estiver configurada ou não houver conexão com a Internet.

Por fim, o processo de sincronização permite que a ferramenta mantenha os dados criptografados na nuvem sempre sincronizados com os dados legíveis que estão armazenados em um diretório na máquina do usuário, de modo que o somente o usuário da ferramenta tenha acesso aos seus dados legíveis.

# 5 CONCLUSÃO

A falta de confidencialidade em nuvens de armazenamento foi identificada através de pesquisas como o trabalho de ROCHA (2010), que mostram alguns dos problemas de segurança em um ambiente de computação em nuvem e também através de notícias recentes do caso da NSA, que mantém programas de vigilância sobre dados que trafegam através da rede mundial de computadores.

Devido a essa falta de confidencialidade foi implementada a ferramenta AntiNSA e o aplicativo KeyManager, utilizando a abordagem de criptografar os dados na máquina do cliente de armazenamento em nuvem, antes que os mesmos sejam enviados à nuvem e permitir que o próprio usuário faça o gerenciamento das chaves utilizadas pela ferramenta. Dessa forma, a ferramenta permite que os dados dos usuários sejam armazenados na nuvem de forma mais segura, aumentando a confiança dos usuários que utilizam os serviços de armazenamento em nuvem, sem que a disponibilidade das informações sejam afetadas, pois um grande diferencial do trabalho foi a utilização de um dispositivo móvel para armazenar as chaves criptográficas através do aplicativo keyManager.

No entanto, o software possui algumas limitações, e como trabalhos futuros pretende-se implementar os seguintes itens: a sincronização de diretórios, pois a ferramenta foi implementada para sincronizar somente arquivos; o compartilhamento de diretórios e arquivos com outros usuários do serviço de armazenamento em nuvem, fazendo o uso de algoritmos de chave assimétrica; implementar melhorias no módulo de segurança inserindo outros algoritmos de criptografia, além do AES; utilizar técnicas de criptografia para armazenar a chave criptográfica no aplicativo KeyManager; implementar outras nuvens de armazenamento como Dropbox e SkyDrive para aumentar a disponibilidade das informações do usuário do serviço, pois a arquitetura implementada foi idealizada para utilizar somente uma nuvem de armazenamento; e adaptar a ferramenta para que a mesma possa ser utilizada em outros sistemas operacionais, como Microsoft Windows[33] e OSx[34], pois a ferramenta AntiNSA utiliza a estrutura de diretórios do

33 http://windows.microsoft.com/en-us/windows/home
34 http://www.apple.com/br/osx/

sistema operacional Linux[35] para armazenar suas configurações.

A abordagem de utilizar técnicas de criptografia na máquina do cliente, no qual o próprio cliente faz o gerenciamento das chaves utilizadas no processo de criptografia, pode ser considerada uma solução eficaz para minimizar a falta de confidencialidade dos serviços de armazenamento em nuvem, dessa forma, o dono das informações será a única pessoa que possuirá acesso aos dados em texto puro.

Por fim, o presente trabalho pode ser uma alternativa para fornecer um nível maior de segurança para o usuário do armazenamento em nuvem, no qual o usuário pode utilizar os serviços de armazenamento em nuvem de forma um pouco mais segura, fazendo o uso da ferramenta AntiNSA e do aplicativo KeyManager que permite manter as chaves criptográficas sempre disponíveis ao usuário.

35 http://www.linux.org/

# REFERÊNCIAS


ARAUJO, Roberto Samarone dos Santos. **GT-Computação em Nuvem para Ciência: Armazenamento de Dados**. UFPA, ago. 2011. Disponível em: <http://www.rnp.br/_arquivo/gt/2011/GT-CNC_f1.pdf>. Acesso em: 27 fev. 2013.

CASTRO, Rita de Cássia de; DOMINGOS, Luiza; LUZ, Glaucivânia; PEREIRA Claudio; GOMES, Marcelo. **Gestão de Vulnerabilidades em Cloud Computing: Um Cenário da Nuvem Pública**. InfoBrasil, mar. 2012. Disponível em: <http://www.infobrasil.inf.br/userfiles/16-S1-2-97170-Gestão%20de%20Vulnerabilidades__.pdf>. Acesso em: 15 fev. 2013.

CHEN, Min-Yu; LIU, Chi-Wei; HWANG, Min-Shiang. **SecureDropbox: A File Encryptiom System Suitable for Cloud Storage Services**. The International Conference on Cloud and Autonomic Computing (CAC 2013), New York, ago. 2013.

EFF - Eletronic Frountier Foundation. **How the NSA's Domestic Spying Program Works**. Disponível em: <https://www.eff.org/nsa-spying/how-it-works>. Acesso em 21 fev. 2013.

G1 - Portal de notícias da Globo. **Entenda o caso de Edward Snowden, que revelou espionagem dos EUA.** São Paulo, fev. 2014. Disponível em: <http://g1.globo.com/mundo/noticia/2013/07/entenda-o-caso-de-edward-snowden-que-revelou-espionagem-dos-eua.html>. Acesso em 23 fev. 2014.

GELLMAN, Barton; SOLTANI, Ashkan. **NSA infiltrates links to Yahoo, Google data centers worldwide, Snowden documents say.** Washington, out. 2013. Disponível em: <http://www.washingtonpost.com/world/national-security/nsa-infiltrates-links-to-yahoo-google-data-centers-worldwide-snowden-documents-say/2013/10/30/e51d661e-4166-11e3-8b74-d89d714ca4dd_story.html>. Acesso em 10 fev. 2014.

GREENWALD, Glenn; MACASKILL, Ewen. **NSA Prism program taps in to user data of Apple, Google and others**. The Guardian. Reino Unido, jun. 2013. Disponível em: <http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data>. Acesso em 15 jan. 2014.

JONES, M. Tim. **Anatomy of a cloud storage infrastructure.** IBM, 2011. Disponível em:<http://www.ibm.com/developerworks/cloud/library/cl-cloudstorage/>. Acesso em: 21 mar. 2013.

KUMAR, Arjun; LEE, Byung Gook; LEE, HoonJae. **Secure Storage and Access of Data in Cloud Computing.** ICT Convergence (ICTC), Jeju Islan, p. 336-339, out. 2012.

MACHADO, Javam C.; SOUZA, Flávio R. C.; MOREIRA, Leonardo O. **Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios**. Universidade Federal do Ceará, 2009. Disponível em: <http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/minicurso/mc7.pdf>. Acesso em: 23 mar. 2013.

MELL, Peter; GRANCE, Timothy. **The NIST Definition of Cloud Computing**. NIST, 2011. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>. Acesso em: 23 jan. de 2013.

NAKAMURA, Emilio Tissato; GEUS, Paulo Lício de. **Segurança de redes em ambientes cooperativos**. São Paulo: Novatec Editora, 2007.

OAUTH - Open standard for authorization. **The OAuth 2.0 Authorization Framework.** Internet Engineering Task Force(IETF), 2012. Disponível em <http://tools.ietf.org/html/rfc6749>. Acesso em 17 mar. de 2014.

PEIXOTO, Mário César Pintaudi. **Engenharia Social e Segurança da Informação na Gestão Corporativa**. 1. ed. Rio de Janeiro: Brasport, 2006.

ROCHA, Francisco Emanuel Liberal. **Privacy in Cloud Computing.** 2010**.** 89f. Dissertação (Mestrado em segurança informática) – Faculdade de Ciências, Universidade de Lisboa, Lisboa, 2010.

SILVA, Fabrício Rodrigues Henriques da. **Um estudo sobre os benefícios e os riscos de segurança na utilização de cloud computing.** 2010. 15 f. Monografia (Artigo científico de conclusão de curso), UNISUAM, 2010. Dísponivel em: <http://fabriciorhs.files.wordpress.com/2011/03/cloud_computing.pdf>. Acesso em: 15 abr. 2013.

SILVA, Felipe Leite da; SILVA, Lucas Melo; ARAÚJO, Roberto. **Aspectos de Segurança: em Nuvens de Armazenamento.** ERN, Guamá, nov. 2012. Disponível em: <http://www.ern2012.com.br/ern2012/images/media/artigo3_ern2012.pdf>. Acesso em: 25 mar. 2013.

STALLINGS, Willian. **Criptografia e segurança de redes**. 4. ed. São Paulo: Pearson Prentice Hall, 2008.

TANENBAUM, Andrew S. **Sistemas operacionais modernos**. 3. ed. São Paulo: Pearson Prentice Hall, 2009.

THE WASHINGTON POST. **NSA slides explain the PRISM data-collection program**. Washington, jul. 2013. Disponível em: <http://www.washingtonpost.com/wp-srv/special/politics/prism-collection-documents>. Acesso em 23 fev. 2014.

VAQUERO, Luis M.; RODERO-MERINO, Luis; CACERES, Juan; LINDNER, Maik. **A Break in the Clouds: Towards a Cloud Definition.** ACM SIGCOMM Computer Communication Review, New York, v. 39, p. 50-55, jan. 2009.

WELLE, Deutsche. **Snowden afirma que NSA espiona por motivos comerciais**. Carta Capital. São Paulo, jan. 2014. Disponível em: <http://www.cartacapital.com.br/internacional/snowden-afirma-que-nsa-espiona-por-motivos-comerciais-8142.html>. Acesso em 15 fev. 2014.

ZHU, Youchan. **A safety design of cloud storage**. Computational and Information Sciences (ICCIS), Chongqing, p. 1054-1057, ago. 2012.

## APÊNDICE A – CÓDIGO FONTE DA FERRAMENTA ANTINSA

```
package br.edu.utfpr.cm.antinsa.configuration;
import br.edu.utfpr.cm.antinsa.dao.TransactionManager;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.SQLException;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import org.jdom2.output.XMLOutputter;

public class Config {

    public static final File STORE_DEFAULT = new File(System.getProperty("user.home"),
"/AntiNSA");
    public static final File STORE_CONFIG = new File(System.getProperty("user.home"),
".antiNSA");
    public static final File XML_CONFIG = new File(STORE_CONFIG + "/config.xml");
    private static Element config;
    private static Document document;

    public static void setup() throws JDOMException, IOException,
ClassNotFoundException, SQLException {
        if (!STORE_DEFAULT.exists()) {
            STORE_DEFAULT.mkdirs();
        }
        if (!STORE_CONFIG.exists()) {
            STORE_CONFIG.mkdirs();
        }
        if (!XML_CONFIG.exists()) {
            createXMLConfig();
        }
        startXMLConfig();
        TransactionManager tm = new TransactionManager();
        tm.createTable();
    }

    private static void startXMLConfig() {
        if (XML_CONFIG.exists()) {
            try {
```

```
            SAXBuilder sb = new SAXBuilder();
            document = sb.build(XML_CONFIG);
            config = document.getRootElement();
        } catch (JDOMException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    } else {
        createXMLConfig();
    }
}

public static void saveXMLConfig() {
    try {
        XMLOutputter xout = new XMLOutputter();
        xout.output(document, new FileWriter(XML_CONFIG));
    } catch (IOException ex) {
        Logger.getLogger(Config.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public static Element readXMLConfig(String tag) {
    List elements = config.getChildren();
    Iterator i = elements.iterator();
    while (i.hasNext()) {
        Element element = (Element) i.next();
        if (element.getName().equals(tag)) {
            return element;
        }
        if (element.getChild(tag) != null) {
            return element.getChild(tag);
        }
    }
    return null;
}

private static void createXMLConfig() {
    config = new Element("config");
    Element googleDrive = new Element("google-drive");
    Element enable = new Element("enable-google-drive");
    Element enableAPIPersonal = new Element("enable-api-personal");
    Element googleName = new Element("google-name");
    Element googleEmail = new Element("google-email");
    Element folderId = new Element("folder-id");
    enable.setText("false");
    enableAPIPersonal.setText("false");
    googleDrive.addContent(enable);
```

```
        googleDrive.addContent(enableAPIPersonal);
        googleDrive.addContent(googleName);
        googleDrive.addContent(googleEmail);
        googleDrive.addContent(folderId);
        config.addContent(googleDrive);
        document = new Document(config);
        saveXMLConfig();
    }

    public static boolean deleteDir(File dir) {
        if (dir.isDirectory()) {
            String[] children = dir.list();
            for (int i = 0; i < children.length; i++) {
                new File(dir + "/" + children[i]).delete();
            }
        }
        return dir.delete();
    }
}

package br.edu.utfpr.cm.antinsa.configuration;
import com.google.api.services.drive.DriveScopes;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.StringReader;
import java.util.Arrays;
import java.util.List;
import org.json.JSONObject;

public class GDUtils {

    public static final File SECRET_KEY = new File(Config.STORE_CONFIG +
"/antinsa.keystore");
    public static final File STORE_CONFIG_GOOGLE_DRIVE = new
File(Config.STORE_CONFIG + "/googledrive");
    public static final File CLIENT_SECRETS = new
File(STORE_CONFIG_GOOGLE_DRIVE + "/client_secrets.json");
    public static final File CACHE_DIR = new File(Config.STORE_DEFAULT + "/.cache");
    public static final String APPLICATION_NAME = "APPJUNIOR";
    public static final String DEFAULT_FOLDER_NAME = "AntiNSA";
    public static final String CLIENT_ID = "440187793751.apps.googleusercontent.com";
    public static final String CLIENT_SECRET = "vmDcytBykuO0UZdxtKlv983g";
    public static final String URL_SERVICE = "https://drive.google.com";
    public static final List<String> SCOPES = Arrays.asList(
```

```
            "https://www.googleapis.com/auth/userinfo.profile",
            "https://www.googleapis.com/auth/userinfo.email",
            DriveScopes.DRIVE_FILE,
            DriveScopes.DRIVE,
            DriveScopes.DRIVE_APPDATA);

    public static void createClientSecrets() throws IOException {
        if (!STORE_CONFIG_GOOGLE_DRIVE.exists()) {
            STORE_CONFIG_GOOGLE_DRIVE.mkdirs();
        }
        if (!CLIENT_SECRETS.exists()) {
            CLIENT_SECRETS.createNewFile();
        }
        try (FileWriter writer = new FileWriter(CLIENT_SECRETS)) {
            JSONObject clientSecrets = new JSONObject();
            clientSecrets.put("client_secret", CLIENT_SECRET);
            clientSecrets.put("client_id", CLIENT_ID);
            JSONObject installed = new JSONObject();
            installed.put("installed", clientSecrets);
            writer.write(installed.toString());
            writer.flush();
            writer.close();
        }
    }

    public static void saveKeyToFile(String key) {

        try {
            FileOutputStream out = new FileOutputStream("/home/junior/antinsa.keystore");
            BufferedReader c = new BufferedReader(new StringReader(key));
            int b;
            while ((b = c.read()) > -1) {
                out.write(b);
            }
            c.close();
            out.close();
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}

package br.edu.utfpr.cm.antinsa.dao;
import br.edu.utfpr.cm.antinsa.googledrive.DataFile;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class DaoDataFile {
    private TransactionManager tm;
    private Statement stmt;

    public DaoDataFile() throws ClassNotFoundException, SQLException {
        initTransactionManager();
    }

    public void insert(String name, long date, String localHash, String cloudHash) throws
SQLException, ClassNotFoundException {
        String sql = "INSERT INTO DATAFILE (NAME,DATE,LOCAL_HASH, CLOUD_HASH)
"
                + "VALUES ('" + name + "', " + date + ", '" + localHash + "', '" + cloudHash + "');";
        stmt.executeUpdate(sql);
    }

    public void update(String name,long date, String localHash, String cloudHash) throws
SQLException {
        String sql = "UPDATE DATAFILE SET NAME = '" + name + "', DATE = " + date + ",
LOCAL_HASH = '" + localHash + "' , CLOUD_HASH = '" + cloudHash + "' WHERE NAME
LIKE '" + name + "';";
        stmt.executeUpdate(sql);
    }

    public void deleteAll() throws SQLException {
        String sql = "DELETE FROM DATAFILE;";
        stmt.executeUpdate(sql);
    }

    public void delete(String fileName) throws SQLException {
        String sql = "DELETE FROM DATAFILE WHERE NAME LIKE '" + fileName + "';";
        stmt.executeUpdate(sql);
    }

    public DataFile getDataFile(String fileName) throws SQLException,
ClassNotFoundException {
        DataFile dataFile = null;
        ResultSet rs = stmt.executeQuery("SELECT * FROM DATAFILE WHERE NAME
LIKE '" + fileName + "';");
        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            long date = rs.getInt("date");
            String localHash = rs.getString("local_hash");
```

```
            String cloudHash = rs.getString("cloud_hash");
            dataFile = new DataFile(id, name, date, localHash, cloudHash);
        }
        rs.close();
        return dataFile;
    }

    public List<DataFile> listAll() throws SQLException, ClassNotFoundException {
        List<DataFile> list = new ArrayList<>();
        ResultSet rs = stmt.executeQuery("SELECT * FROM DATAFILE;");
        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            long date = rs.getLong("date");
            String localHash = rs.getString("local_hash");
            String cloudHash = rs.getString("cloud_hash");
            list.add(new DataFile(id, name, date, localHash, cloudHash));
        }
        rs.close();
        return list;
    }

    private void initTransactionManager() throws SQLException, ClassNotFoundException {
        tm = new TransactionManager();
        stmt = tm.getStatement();
    }

    public TransactionManager getTransactionManager() {
        return tm;
    }

    public boolean dataFileExists(String fileName) throws SQLException,
ClassNotFoundException {
        ResultSet rs = stmt.executeQuery("SELECT * FROM DATAFILE WHERE NAME
LIKE '" + fileName + "';");
        return rs.next();
    }
}

package br.edu.utfpr.cm.antinsa.dao;
import br.edu.utfpr.cm.antinsa.configuration.Config;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class TransactionManager {
    private Connection connection = null;
```

```
    private Statement stmt;

    public Connection getConnection() throws ClassNotFoundException, SQLException {
        if (connection == null) {
            Class.forName("org.sqlite.JDBC");
            connection = DriverManager.getConnection("jdbc:sqlite:" +
Config.STORE_CONFIG + "/antinsa.db");
            stmt = (Statement) connection.createStatement();
        }
        return connection;
    }

    public void createTable() throws ClassNotFoundException, SQLException {
        getConnection();
        String sql = "CREATE TABLE IF NOT EXISTS DATAFILE "
                + "(ID INTEGER PRIMARY KEY AUTOINCREMENT  NOT NULL,"
                + " NAME          TEXT UNIQUE   NOT NULL, "
                + " DATE          LONG     NOT NULL, "
                + " LOCAL_HASH        TEXT , "
                + " CLOUD_HASH        TEXT )";
        stmt.executeUpdate(sql);
        close();
    }

    public Statement getStatement() throws ClassNotFoundException, SQLException {
        if (stmt == null) {
            stmt = getConnection().createStatement();
        }
        return stmt;
    }

    public void close() throws SQLException {
        stmt.close();
        connection.close();
    }

    public void commit() throws SQLException {
        connection.commit();
        close();
    }
}

package br.edu.utfpr.cm.antinsa.googledrive;

public class DataFile {
    private int id;
    private String name;
    private long date;
```

```
    private String hash;
    private String cloudHash;

    public DataFile(int id, String name,long date, String hash, String cloudHash) {
        this.id = id;
        this.name = name;
        this.date = date;
        this.hash = hash;
        this.cloudHash = cloudHash;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }

    public long getDate() {
        return date;
    }

    public void setDate(long date) {
        this.date = date;
    }

    public String getCloudHash() {
        return cloudHash;
    }

    public void setCloudHash(String cloudHash) {
        this.cloudHash = cloudHash;
    }
}
```

```
package br.edu.utfpr.cm.antinsa.googledrive;
import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import br.edu.utfpr.cm.antinsa.configuration.Config;
import br.edu.utfpr.cm.antinsa.util.Util;
import com.google.api.client.auth.oauth2.Credential;
import com.google.api.client.http.FileContent;
import com.google.api.client.http.GenericUrl;
import com.google.api.client.http.HttpResponse;
import com.google.api.client.http.HttpTransport;
import com.google.api.client.http.javanet.NetHttpTransport;
import com.google.api.client.json.JsonFactory;
import com.google.api.client.json.jackson2.JacksonFactory;
import com.google.api.client.util.DateTime;
import com.google.api.services.drive.Drive;
import com.google.api.services.drive.Drive.Files;
import com.google.api.services.drive.model.About;
import com.google.api.services.drive.model.File;
import com.google.api.services.drive.model.FileList;
import com.google.api.services.drive.model.ParentReference;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.security.GeneralSecurityException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import javax.swing.JOptionPane;

public class GoogleDrive {
    private Credential credential;
    private HttpTransport httpTransport;
    private JsonFactory jsonFactory;
    private Drive service;

    public GoogleDrive() throws IOException, GeneralSecurityException {
        buildGoogleDriveService();
        verifyDefaultFolder();
    }

    private void buildGoogleDriveService() throws IOException, GeneralSecurityException {
        if (service == null) {
            httpTransport = new NetHttpTransport();
            jsonFactory = new JacksonFactory();
            credential = GoogleDriveOAuth.getCredential();
            service = new Drive.Builder(httpTransport, jsonFactory,
credential).setApplicationName(GDUtils.APPLICATION_NAME).build();
```

```
        }
    }

    public InputStream downloadFile(String downloadUrl) {
        if (Util.verifyServiceConnection(GDUtils.URL_SERVICE)) {
            if (downloadUrl != null && downloadUrl.length() > 0) {
                try {
                    HttpResponse resp =
                            service.getRequestFactory().buildGetRequest(new
GenericUrl(downloadUrl))
                                    .execute();
                    return resp.getContent();
                } catch (IOException e) {
                    e.printStackTrace();
                    return null;
                }
            }
        }
        return null;
    }

    public List<File> getFilesDefaultFolder() throws IOException, GeneralSecurityException
{
        List<File> result = new ArrayList<File>();
        String parentId = Config.readXMLConfig("folder-id").getText();
        Files.List request = service.files().list().setQ(" '" + parentId + "' in parents  and
trashed=false");
        do {
            try {
                FileList files = request.execute();
                result.addAll(files.getItems());
                request.setPageToken(files.getNextPageToken());
            } catch (IOException e) {
                System.out.println("An error occurred: " + e);
                request.setPageToken(null);
            }
        } while (request.getPageToken() != null
                && request.getPageToken().length() > 0);
        return result;
    }

    private void createDefaultFolder() {
        try {
            File body = new File();
            body.setTitle(GDUtils.DEFAULT_FOLDER_NAME);
            body.setMimeType("application/vnd.google-apps.folder");
            File folder = service.files().insert(body).execute();
            Config.readXMLConfig("folder-id").setText(folder.getId());
```

```
            Config.saveXMLConfig();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private void verifyDefaultFolder() {
        try {
            Files.List request = service.files().list().setQ(
                    "mimeType='application/vnd.google-apps.folder' and trashed=false");
            FileList folders = request.execute();
            for (File folder : folders.getItems()) {
                if (folder.getTitle().equals(GDUtils.DEFAULT_FOLDER_NAME)) {
                    Config.readXMLConfig("folder-id").setText(folder.getId());
                    Config.saveXMLConfig();
                    return;
                }
            }
            createDefaultFolder();
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
        }
    }

    private String getFolderId() {
        try {
            Files.List request = service.files().list().setQ(
                    "mimeType='application/vnd.google-apps.folder' and trashed=false");
            FileList files = request.execute();
            for (File file : files.getItems()) {
                if (file.getTitle().equals(GDUtils.DEFAULT_FOLDER_NAME)) {
                    return file.getId();
                }
            }
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
        }
        return null;
    }

    public File getFileId(String filename) {
        try {
            FileList files = service.files().list().execute();
            String parentId = Config.readXMLConfig("folder-id").getText();
            for (File file : files.getItems()) {
                if (file.getTitle().equals(filename)) {
```

```
                for (ParentReference parent : file.getParents()) {
                    if (parent.getId().equals(parentId)) {
                        return file;
                    }
                }
            }
        }
    } catch (IOException ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
    }
    return null;
  }

  public File createFile(java.io.File file, long lastModified) {
    if (Util.verifyServiceConnection(GDUtils.URL_SERVICE)) {
        if (hasStorage(file.length())) {
            try {
                String parentId = Config.readXMLConfig("folder-id").getText();
                File body = new File();
                body.setTitle(file.getName());
                body.setModifiedDate(new DateTime(lastModified));
                if (parentId == null) {
                    verifyDefaultFolder();
                } else {
                    body.setParents(
                            Arrays.asList(new ParentReference().setId(parentId)));
                }

                java.io.File fileContent = new java.io.File(file.getAbsolutePath());

                FileContent mediaContent = new
FileContent(Util.getMimeType(file.getAbsolutePath()), fileContent);
                File fileCloud = service.files().insert(body, mediaContent).execute();
                return fileCloud;

            } catch (IOException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(null, "The user quota has been exceeded!",
"WARNING", JOptionPane.WARNING_MESSAGE);
        }
    }
    return null;
  }
```

```
    public boolean deleteFile(String fileName) {
        if (Util.verifyServiceConnection(GDUtils.URL_SERVICE)) {
            try {
                File file = getFileId(fileName);
                if (file != null) {
                    buildGoogleDriveService();
                    service.files().delete(file.getId()).execute();
                    return true;
                }
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
            } catch (GeneralSecurityException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
            }
        }
        return false;
    }

    public File updateFile(java.io.File file, long lastModified) {
        if (Util.verifyServiceConnection(GDUtils.URL_SERVICE)) {
            if (hasStorage(file.length())) {
                try {
                    File fileCloud = getFileId(file.getName());
                    if (fileCloud != null) {
                        String parentId = Config.readXMLConfig("folder-id").getText();
                        File body = new File();
                        body.setTitle(file.getName());
                        body.setModifiedDate(new DateTime(lastModified));
                        if (parentId == null) {
                            verifyDefaultFolder();
                        } else {
                            body.setParents(
                                    Arrays.asList(new ParentReference().setId(parentId)));
                        }
                        FileContent mediaContent = new
FileContent(Util.getMimeType(file.getAbsolutePath()), file);
                        return service.files().update(fileCloud.getId(), body,
mediaContent).execute();
                    }
                } catch (IOException ex) {
                    JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
                }
            } else {
                JOptionPane.showMessageDialog(null, "The user quota has been exceeded!",
"WARNING", JOptionPane.WARNING_MESSAGE);
```

```
            }
        }
        return null;
    }

    public java.io.File saveFile(InputStream input, String name) throws
FileNotFoundException, IOException {
        if (input != null) {
            java.io.File file = new java.io.File(GDUtils.CACHE_DIR + "/" + name);
            FileOutputStream out = new FileOutputStream(file);
            int b;
            while ((b = input.read()) > -1) {
                out.write(b);
            }
            input.close();
            out.close();
            return file;
        }
        return null;
    }
    private boolean hasStorage(long sizeFile) {
        try {
            About about = service.about().get().execute();
            Long used = (about.getQuotaBytesUsed() + sizeFile);
            if (about.getQuotaBytesTotal() >= used) {
                return true;
            }
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
        }
        return false;
    }
}

package br.edu.utfpr.cm.antinsa.googledrive;
import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import br.edu.utfpr.cm.antinsa.configuration.Config;
import br.edu.utfpr.cm.antinsa.dao.DaoDataFile;
import br.edu.utfpr.cm.antinsa.security.SecretKeyAESCrypto;
import br.edu.utfpr.cm.antinsa.security.HashGenerator;
import br.edu.utfpr.cm.antinsa.util.Util;
import com.google.api.services.drive.model.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.security.GeneralSecurityException;
import java.sql.SQLException;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Scanner;
import javax.swing.JOptionPane;

public class GoogleDriveController extends Thread {
    private DaoDataFile daoDataFile;
    private java.io.File[] files;
    private List<DataFile> dbFiles;
    private java.io.File dir;
    private GoogleDrive googleDrive;
    private List<File> cloudFiles;
    private java.io.File encryptedFile;
    private java.io.File decryptedFile;
    private SecretKeyAESCrypto cipher;
    private boolean isClosed = true;

    public GoogleDriveController() {
        try {
            daoDataFile = new DaoDataFile();
            dir = new java.io.File(Config.STORE_DEFAULT.getAbsolutePath());
            googleDrive = new GoogleDrive();
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        } catch (SQLException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (GeneralSecurityException ex) {
            ex.printStackTrace();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public void run() {
        try {
            while (!isClosed) {
                if (cipher == null) {
                    if (GDUtils.SECRET_KEY.exists()) {
                        cipher = new SecretKeyAESCrypto();
                    }
                }
                if (Config.STORE_CONFIG.exists() &&
Util.verifyServiceConnection(GDUtils.URL_SERVICE) && GDUtils.SECRET_KEY.exists())
```

```
{
                Config.STORE_DEFAULT.mkdirs();
                GDUtils.CACHE_DIR.mkdirs();
                cloudSync();
                localSync();
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }

}

public void initServiceGoogleDrive() {
    try {
        if (Config.readXMLConfig("enable-google-drive").getText().equals("true") && !
GoogleDriveOAuth.isValidCredential()) {
            try {
                int value = JOptionPane.showConfirmDialog(null, "Your Google Drive account
is enable, but isn't authorized! \n Would you like to open your browser to perform
authorization?", "Information", JOptionPane.YES_NO_OPTION);
                if (value == JOptionPane.YES_OPTION) {
                    GoogleDriveOAuth.getCredential();
                    JOptionPane.showMessageDialog(null, "Authentication performed
successfully!", "Sucessful", JOptionPane.INFORMATION_MESSAGE);
                    start();
                } else {
                    Config.readXMLConfig("enable-google-drive").setText("false");
                    Config.saveXMLConfig();
                }
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
                ex.printStackTrace();
            } catch (GeneralSecurityException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
                ex.printStackTrace();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
                ex.printStackTrace();
            }
        } else if (Config.readXMLConfig("enable-google-drive").getText().equals("true") &&
GoogleDriveOAuth.isValidCredential()) {
            try {
                GoogleDriveOAuth.getCredential();
                start();
```

```
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
                ex.printStackTrace();
            } catch (GeneralSecurityException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
                ex.printStackTrace();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
                ex.printStackTrace();
            }
        }
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
        ex.printStackTrace();
    }
}

private boolean isTempFile(String fileName) {
    char c1 = fileName.charAt(fileName.length() - 1);
    char c2 = fileName.charAt(0);
    if (c1 != '~' && c2 != '.') {
        return false;
    }
    return true;
}

private void localSync() {
    try {
        files = dir.listFiles();
        dbFiles = daoDataFile.listAll();
        for (java.io.File file : files) {
            if (!isTempFile(file.getAbsolutePath()) && !
Util.getMimeType(file.getAbsolutePath()).equals("inode/directory") && !
isOpen(file.getAbsolutePath().replace(" ", "\\ "))) {
                DataFile dataFile = daoDataFile.getDataFile(file.getName());
                if (dataFile != null) {
                    if (file.lastModified() > dataFile.getDate()) {
                        String hash = HashGenerator.hashFile(file.getAbsolutePath());
                        if (!dataFile.getHash().equals(hash)) {
                            encryptedFile = cipher.encrypt(file);
                            File fileUpdated = googleDrive.updateFile(encryptedFile,
file.lastModified());
                            if (fileUpdated != null) {
                                daoDataFile.update(file.getName(), file.lastModified(),
```

```
HashGenerator.hashFile(file.getAbsolutePath()), fileUpdated.getMd5Checksum());
                        }
                    }
                }
            } else {
                encryptedFile = cipher.encrypt(file);
                File fileCreated = googleDrive.createFile(encryptedFile, file.lastModified());
                if (fileCreated != null) {
                    daoDataFile.insert(file.getName(), file.lastModified(),
HashGenerator.hashFile(file.getAbsolutePath()), fileCreated.getMd5Checksum());
                }
            }
        }
        if (encryptedFile != null) {
            encryptedFile.delete();
        }
    }
    for (DataFile dataFile : dbFiles) {
        int count = 0;
        for (java.io.File file : files) {
            if (dataFile.getName().equals(file.getName())) {
                count = 1;
            }
        }
        if (count == 0) {
            if (googleDrive.deleteFile(dataFile.getName())) {
                daoDataFile.delete(dataFile.getName());
            }
        }
    }
  } catch (Exception ex) {
    ex.printStackTrace();
  }
}

private void cloudSync() {
  try {
    dbFiles = daoDataFile.listAll();
    cloudFiles = googleDrive.getFilesDefaultFolder();
    for (File file : cloudFiles) {
        DataFile dataFile = daoDataFile.getDataFile(file.getTitle());
        if (dataFile != null) {
            if (dataFile.getName().equals(file.getTitle()) && dataFile.getDate() <
file.getModifiedByMeDate().getValue()) {
                if (!dataFile.getCloudHash().equals(file.getMd5Checksum())) {
                    encryptedFile =
googleDrive.saveFile(googleDrive.downloadFile(file.getDownloadUrl()), file.getTitle());
                    if (encryptedFile != null) {
```

```
                        decryptedFile = cipher.decrypt(encryptedFile);

decryptedFile.setLastModified(file.getLastViewedByMeDate().getValue());
                        daoDataFile.update(decryptedFile.getName(),
decryptedFile.lastModified(), HashGenerator.hashFile(decryptedFile.getAbsolutePath()),
file.getMd5Checksum());
                        encryptedFile.delete();
                    }
                }
            }
        } else {
            if (file.getDownloadUrl() != null) {
                encryptedFile =
googleDrive.saveFile(googleDrive.downloadFile(file.getDownloadUrl()), file.getTitle());
                if (encryptedFile != null) {
                    decryptedFile = cipher.decrypt(encryptedFile);

decryptedFile.setLastModified(file.getLastViewedByMeDate().getValue());
                    daoDataFile.insert(decryptedFile.getName(),
decryptedFile.lastModified(), HashGenerator.hashFile(decryptedFile.getAbsolutePath()),
file.getMd5Checksum());
                    encryptedFile.delete();
                }
            }
        }
        for (DataFile dataFile : dbFiles) {
            int count = 0;
            for (File file : cloudFiles) {
                if (dataFile.getName().equals(file.getTitle())) {
                    count = 1;
                }
            }
            if (count == 0) {
                daoDataFile.delete(dataFile.getName());
                new java.io.File(Config.STORE_DEFAULT.getAbsolutePath() + "/" +
dataFile.getName()).delete();
            }
        }

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

private void saveFile(java.io.File file) throws FileNotFoundException, IOException {
    FileOutputStream out = new FileOutputStream(new
java.io.File(Config.STORE_DEFAULT + file.getName()));
```

```
        InputStream input = new FileInputStream(file);
        int b;
        while ((b = input.read()) > -1) {
            out.write(b);
        }
        input.close();
        out.close();
    }

    public boolean isOpen(String path) {
        try {
            Runtime r = Runtime.getRuntime();
            Process p = r.exec("lsof " + path);
            Scanner scanner = new Scanner(p.getInputStream());
            scanner.useDelimiter("$$").next();
            return true;
        } catch (NoSuchElementException ex) {
            return false;
        } catch (IOException ex) {
            return false;
        }
    }

    @Override
    public void start() {
        isClosed = false;
        super.start();
    }

    public void close() {
        isClosed = true;
    }
}
```

```
package br.edu.utfpr.cm.antinsa.googledrive;
import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import br.edu.utfpr.cm.antinsa.configuration.Config;
import br.edu.utfpr.cm.antinsa.util.Util;
import com.google.api.client.auth.oauth2.Credential;
import com.google.api.client.auth.oauth2.TokenResponse;
import com.google.api.client.extensions.java6.auth.oauth2.AuthorizationCodeInstalledApp;
import com.google.api.client.extensions.jetty.auth.oauth2.LocalServerReceiver;
import com.google.api.client.googleapis.auth.oauth2.GoogleAuthorizationCodeFlow;
import com.google.api.client.googleapis.auth.oauth2.GoogleClientSecrets;
import com.google.api.client.googleapis.javanet.GoogleNetHttpTransport;
import com.google.api.client.http.HttpTransport;
import com.google.api.client.json.JsonFactory;
import com.google.api.client.json.jackson2.JacksonFactory;
```

```
import com.google.api.client.util.store.FileDataStoreFactory;
import com.google.api.services.oauth2.Oauth2;
import com.google.api.services.oauth2.model.Tokeninfo;
import com.google.api.services.oauth2.model.Userinfo;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.security.GeneralSecurityException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class GoogleDriveOAuth {
    private static FileDataStoreFactory dataStoreFactory;
    private static HttpTransport httpTransport;
    private static final JsonFactory JSON_FACTORY =
JacksonFactory.getDefaultInstance();
    private static Oauth2 oauth2;
    private static Credential credential;
    private static GoogleClientSecrets clientSecrets;
    private static GoogleAuthorizationCodeFlow flow;

    public static Credential authorize() throws IOException, GeneralSecurityException {
        if (!GDUtils.CLIENT_SECRETS.exists()) {
            GDUtils.createClientSecrets();
        }
        clientSecrets = GoogleClientSecrets.load(JSON_FACTORY,
                new InputStreamReader(new FileInputStream(GDUtils.CLIENT_SECRETS)));
        httpTransport = GoogleNetHttpTransport.newTrustedTransport();
        dataStoreFactory = new
FileDataStoreFactory(GDUtils.STORE_CONFIG_GOOGLE_DRIVE);
        flow = new GoogleAuthorizationCodeFlow.Builder(
                httpTransport, JSON_FACTORY, clientSecrets,
GDUtils.SCOPES).setDataStoreFactory(
                dataStoreFactory).build();
        credential = new AuthorizationCodeInstalledApp(flow, new
LocalServerReceiver()).authorize("user");
        if (Config.readXMLConfig("google-name").getText().equals("")) {
            saveAccountInfo();
        }
        flow.createAndStoreCredential(new
TokenResponse().setAccessToken(credential.getAccessToken()), "1");
        return credential;
    }

    private static void tokenInfo(String accessToken) {
        Tokeninfo tokeninfo;
        try {
            tokeninfo = oauth2.tokeninfo().setAccessToken(accessToken).execute();
```

```
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public static Oauth2 getOauth2() throws IOException, GeneralSecurityException {
        if (oauth2 == null) {
            getCredential();
            oauth2 = new Oauth2.Builder(httpTransport, JSON_FACTORY,
credential).setApplicationName(
                    GDUtils.APPLICATION_NAME).build();
            tokenInfo(credential.getAccessToken());
        }
        return oauth2;
    }

    public static Credential getCredential() throws IOException, GeneralSecurityException {
        if (credential == null) {
            authorize();
        }
        return credential;
    }

    public static Userinfo userInfo() throws IOException, GeneralSecurityException {
        oauth2 = getOauth2();
        return oauth2.userinfo().get().execute();
    }

    private static void saveAccountInfo() throws IOException {
        if (Util.verifyServiceConnection(GDUtils.URL_SERVICE)) {
            try {
                Config.readXMLConfig("google-name").setText(userInfo().getName());
                Config.readXMLConfig("google-email").setText(userInfo().getEmail());
                Config.saveXMLConfig();
            } catch (IOException ex) {
                ex.printStackTrace();
            } catch (GeneralSecurityException ex) {
                ex.printStackTrace();
            }
        }
    }

    public static boolean isValidCredential() {
        try {
            if (!GDUtils.CLIENT_SECRETS.exists()) {
                GDUtils.createClientSecrets();
            }
            clientSecrets = GoogleClientSecrets.load(JSON_FACTORY,
```

```
                new InputStreamReader(new FileInputStream(GDUtils.CLIENT_SECRETS)));
            try {
                httpTransport = GoogleNetHttpTransport.newTrustedTransport();
            } catch (GeneralSecurityException ex) {
                Logger.getLogger(GoogleDriveOAuth.class.getName()).log(Level.SEVERE, null,
ex);
            }
            dataStoreFactory = new
FileDataStoreFactory(GDUtils.STORE_CONFIG_GOOGLE_DRIVE);
            flow = new GoogleAuthorizationCodeFlow.Builder(
                    httpTransport, JSON_FACTORY, clientSecrets,
GDUtils.SCOPES).setDataStoreFactory(
                    dataStoreFactory).build();
            Credential loadCredential = flow.loadCredential("1");
            if (loadCredential != null) {
                return true;
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return false;
    }
}

package br.edu.utfpr.cm.antinsa.gui;
import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import br.edu.utfpr.cm.antinsa.security.KeyManager;
import br.edu.utfpr.cm.antinsa.security.SSLSocketClient;
import br.edu.utfpr.cm.antinsa.util.Util;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.ConnectException;
import java.security.GeneralSecurityException;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class JDialogReceiveKey extends javax.swing.JDialog {
    private SSLSocketClient socketClient;

    public JDialogReceiveKey(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        socketClient = new SSLSocketClient();
        jTextFieldPort.setText(String.valueOf(socketClient.getPort()));
        jLabelLocation.setEnabled(jRadioButtonLocal.isSelected());
        jTextFieldFile.setEnabled(jRadioButtonLocal.isSelected());
        jButtonSelect.setEnabled(jRadioButtonLocal.isSelected());
```

```
    }
@SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        buttonGroup1 = new javax.swing.ButtonGroup();
        jPanel1 = new javax.swing.JPanel();
        jRadioButtonSmartphone = new javax.swing.JRadioButton();
        jRadioButtonLocal = new javax.swing.JRadioButton();
        jLabel1 = new javax.swing.JLabel();
        jSeparator1 = new javax.swing.JSeparator();
        jSeparator2 = new javax.swing.JSeparator();
        jLabelLocation = new javax.swing.JLabel();
        jTextFieldFile = new javax.swing.JTextField();
        jButtonSelect = new javax.swing.JButton();
        jLabelIP = new javax.swing.JLabel();
        jTextFieldPort = new javax.swing.JTextField();
        jLabelPort = new javax.swing.JLabel();
        jTextFieldIP = new javax.swing.JTextField();
        jLabelKeyName = new javax.swing.JLabel();
        jTextFieldKeyName = new javax.swing.JTextField();
        jSeparator3 = new javax.swing.JSeparator();
        jSeparator4 = new javax.swing.JSeparator();
        jButton1 = new javax.swing.JButton();
        jToggleButton1 = new javax.swing.JToggleButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

        jPanel1.setToolTipText("Save key");

        buttonGroup1.add(jRadioButtonSmartphone);
        jRadioButtonSmartphone.setSelected(true);
        jRadioButtonSmartphone.setText("Smartphone");
        jRadioButtonSmartphone.addChangeListener(new
javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                jRadioButtonSmartphoneStateChanged(evt);
            }
        });

        buttonGroup1.add(jRadioButtonLocal);
        jRadioButtonLocal.setText("Local directory");
        jRadioButtonLocal.addChangeListener(new javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                jRadioButtonLocalStateChanged(evt);
            }
        });
```

```
jLabel1.setText("Select how to want to get the key:");

jLabelLocation.setText("Location:");

jTextFieldFile.setPreferredSize(new java.awt.Dimension(10, 30));

jButtonSelect.setText("Select");
jButtonSelect.setPreferredSize(new java.awt.Dimension(94, 33));
jButtonSelect.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonSelectActionPerformed(evt);
    }
});

jLabelIP.setText("IP Address:");

jTextFieldPort.setEditable(false);
jTextFieldPort.setPreferredSize(new java.awt.Dimension(84, 30));

jLabelPort.setText("Port:");

jTextFieldIP.setPreferredSize(new java.awt.Dimension(84, 30));

jLabelKeyName.setText("Key name:");

jTextFieldKeyName.setPreferredSize(new java.awt.Dimension(84, 30));

jButton1.setText("Close");
jButton1.setPreferredSize(new java.awt.Dimension(97, 33));
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});

jToggleButton1.setText("Get");
jToggleButton1.setPreferredSize(new java.awt.Dimension(97, 33));
jToggleButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jToggleButton1ActionPerformed(evt);
    }
});

javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
```

```
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAIL
ING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addContainerGap()
                    .addComponent(jSeparator3))
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addGap(49, 49, 49)
                            .addComponent(jRadioButtonSmartphone)
                            .addGap(39, 39, 39)
                            .addComponent(jRadioButtonLocal))
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addContainerGap()
                            .addComponent(jLabel1)))
                    .addGap(0, 0, Short.MAX_VALUE))
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(30, 30, 30)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                        .addComponent(jLabelIP)
                        .addComponent(jLabelLocation)
                        .addComponent(jLabelKeyName))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 27,
Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                        .addComponent(jTextFieldKeyName,
javax.swing.GroupLayout.PREFERRED_SIZE, 445,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addComponent(jTextFieldFile,
javax.swing.GroupLayout.PREFERRED_SIZE, 339,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                            .addComponent(jButtonSelect,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(jPanel1Layout.createSequentialGroup()
```

```
                        .addComponent(jTextFieldIP,
javax.swing.GroupLayout.PREFERRED_SIZE, 240,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(18, 18, 18)
                        .addComponent(jLabelPort)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jTextFieldPort,
javax.swing.GroupLayout.PREFERRED_SIZE, 141,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                    .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                        .addComponent(jSeparator4,
javax.swing.GroupLayout.Alignment.TRAILING)
                        .addComponent(jSeparator1))))
            .addContainerGap())
          .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
            .addGap(0, 0, Short.MAX_VALUE)
            .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(18, 18, 18)
            .addComponent(jToggleButton1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(23, 23, 23))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addGroup(jPanel1Layout.createSequentialGroup()
              .addContainerGap()
              .addComponent(jSeparator2, javax.swing.GroupLayout.DEFAULT_SIZE, 568,
Short.MAX_VALUE)
              .addContainerGap()))
      );
      jPanel1Layout.setVerticalGroup(
         jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(jPanel1Layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jSeparator3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(18, 18, 18)
```

```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                .addComponent(jRadioButtonSmartphone)
                .addComponent(jRadioButtonLocal))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 6,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                .addComponent(jTextFieldFile, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jButtonSelect,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelLocation))
            .addGap(31, 31, 31)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                .addComponent(jTextFieldPort,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelPort)
                .addComponent(jTextFieldIP, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelIP))
            .addGap(25, 25, 25)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                .addComponent(jLabelKeyName)
                .addComponent(jTextFieldKeyName,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jSeparator4, javax.swing.GroupLayout.PREFERRED_SIZE, 12,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
18, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
```

```
                .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 34,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jToggleButton1,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addContainerGap())

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(150, 150, 150)
                .addComponent(jSeparator2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(183, Short.MAX_VALUE)))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(0, 0, Short.MAX_VALUE)
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(0, 0, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(0, 8, Short.MAX_VALUE)
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(0, 8, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        setVisible(false);
    }

    private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        if (jRadioButtonLocal.isSelected()) {
            String path = jTextFieldFile.getText();

            if ((path != null && !"".equals(path)) && !"/".equals(path)) {
                if (KeyManager.isValidKey(path)) {
```

```
                boolean copyKey = copyKey(new File(path));
                if (copyKey) {
                    JOptionPane.showMessageDialog(this, "The key was save with
successfully!", "Sucessful", JOptionPane.INFORMATION_MESSAGE);
                }
            }else{
                JOptionPane.showMessageDialog(this, "Invalid keystore format!", "Warning",
JOptionPane.WARNING_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(this, "You need to select the location!",
"Warning", JOptionPane.WARNING_MESSAGE);
        }
    } else {
        try {
            if (!"".equals(jTextFieldIP.getText())) {
                if (jTextFieldKeyName.getText().length() > 5) {
                    SSLSocketClient client = new SSLSocketClient();
                    Util.createPane(this, "Wait ... Establishing communication with the server!");
                    client.connect(jTextFieldIP.getText());
                    client.sendMessage("2");
                    client.sendMessage(jTextFieldKeyName.getText());
                    client.sendMessage("");
                    String receiveMessage = client.receiveMessage();
                    if (!"0".equals(receiveMessage) && !"".equals(receiveMessage)) {
                        JOptionPane.showMessageDialog(this, "The key will be stored in config
directory!", "Information", JOptionPane.INFORMATION_MESSAGE);
                        try {
                            KeyManager.storeSecretKeyFile(receiveMessage);
                            JOptionPane.showMessageDialog(this, "The key was received with
success", "Information", JOptionPane.INFORMATION_MESSAGE);
                            jTextFieldKeyName.setText("");
                        } catch (IOException ex) {
                            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
                        }
                    } else {
                        if ("0".equals(receiveMessage)) {
                            JOptionPane.showMessageDialog(this, "There isn't exists a key with
the name specified!", "Information", JOptionPane.INFORMATION_MESSAGE);
                        } else {
                            JOptionPane.showMessageDialog(this, "Key not received", "Error",
JOptionPane.WARNING_MESSAGE);
                        }
                    }
                    client.close();
                } else {
                    JOptionPane.showMessageDialog(this, "The field key name must have
```

```
more than 5 characters!", "Information", JOptionPane.INFORMATION_MESSAGE);
                }
            } else {
                JOptionPane.showMessageDialog(this, "You need to insert the IP Address!",
"Information", JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (ConnectException ex) {
            JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!\n" + ex.getMessage(), "Warning", JOptionPane.WARNING_MESSAGE);
        } catch (GeneralSecurityException ex) {
            JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!", "Warning", JOptionPane.WARNING_MESSAGE);
        } catch (FileNotFoundException ex) {
            JOptionPane.showMessageDialog(this, "Key not found", "Warning!",
JOptionPane.WARNING_MESSAGE);
        } catch (ClassNotFoundException ex) {
            JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!", "Warning", JOptionPane.WARNING_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!", "Warning", JOptionPane.WARNING_MESSAGE);
        }
    }
}

private void jRadioButtonSmartphoneStateChanged(javax.swing.event.ChangeEvent
evt) {
    jLabelIP.setEnabled(jRadioButtonSmartphone.isSelected());
    jTextFieldIP.setEnabled(jRadioButtonSmartphone.isSelected());
    jTextFieldPort.setEnabled(jRadioButtonSmartphone.isSelected());
    jLabelPort.setEnabled(jRadioButtonSmartphone.isSelected());
    jTextFieldKeyName.setEnabled(jRadioButtonSmartphone.isSelected());
    jLabelKeyName.setEnabled(jRadioButtonSmartphone.isSelected());
}

private void jRadioButtonLocalStateChanged(javax.swing.event.ChangeEvent evt) {
    jLabelLocation.setEnabled(jRadioButtonLocal.isSelected());
    jTextFieldFile.setEnabled(jRadioButtonLocal.isSelected());
    jButtonSelect.setEnabled(jRadioButtonLocal.isSelected());
}

private void jButtonSelectActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser file = new JFileChooser();
    file.setFileSelectionMode(JFileChooser.FILES_ONLY);
    int i = file.showSaveDialog(null);
    if (i == 1) {
        jTextFieldFile.setText("");
    } else {
```

```
            String path = file.getSelectedFile().getAbsolutePath();
            jTextFieldFile.setText(path);
        }
    }
    // Variables declaration - do not modify
    private javax.swing.ButtonGroup buttonGroup1;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButtonSelect;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabelIP;
    private javax.swing.JLabel jLabelKeyName;
    private javax.swing.JLabel jLabelLocation;
    private javax.swing.JLabel jLabelPort;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JRadioButton jRadioButtonLocal;
    private javax.swing.JRadioButton jRadioButtonSmartphone;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JSeparator jSeparator3;
    private javax.swing.JSeparator jSeparator4;
    private javax.swing.JTextField jTextFieldFile;
    private javax.swing.JTextField jTextFieldIP;
    private javax.swing.JTextField jTextFieldKeyName;
    private javax.swing.JTextField jTextFieldPort;
    private javax.swing.JToggleButton jToggleButton1;
    // End of variables declaration

    public boolean copyKey(File newfile) {
        try {
            Util.copyFile(newfile, GDUtils.SECRET_KEY);
            return true;
        } catch (FileNotFoundException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
        return false;
    }
}

package br.edu.utfpr.cm.antinsa.gui;
import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import br.edu.utfpr.cm.antinsa.security.KeyManager;
```

```
import br.edu.utfpr.cm.antinsa.security.SSLSocketClient;
import br.edu.utfpr.cm.antinsa.util.Util;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.ConnectException;
import java.security.GeneralSecurityException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class JDialogSaveKey extends javax.swing.JDialog {
    private SSLSocketClient socketClient;

    public JDialogSaveKey(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        socketClient = new SSLSocketClient();
        jTextFieldPort.setText(String.valueOf(socketClient.getPort()));
        jLabelLocation.setEnabled(jRadioButtonLocal.isSelected());
        jTextFieldDirectory.setEnabled(jRadioButtonLocal.isSelected());
        jButtonSelect.setEnabled(jRadioButtonLocal.isSelected());
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        buttonGroup1 = new javax.swing.ButtonGroup();
        jPanel1 = new javax.swing.JPanel();
        jRadioButtonSmartphone = new javax.swing.JRadioButton();
        jRadioButtonLocal = new javax.swing.JRadioButton();
        jLabel1 = new javax.swing.JLabel();
        jSeparator1 = new javax.swing.JSeparator();
        jSeparator2 = new javax.swing.JSeparator();
        jLabelLocation = new javax.swing.JLabel();
        jTextFieldDirectory = new javax.swing.JTextField();
        jButtonSelect = new javax.swing.JButton();
        jLabelIP = new javax.swing.JLabel();
        jTextFieldPort = new javax.swing.JTextField();
        jLabelPort = new javax.swing.JLabel();
        jTextFieldIP = new javax.swing.JTextField();
        jLabelKeyName = new javax.swing.JLabel();
        jTextFieldKeyName = new javax.swing.JTextField();
        jSeparator3 = new javax.swing.JSeparator();
        jSeparator4 = new javax.swing.JSeparator();
```

```
        jButton1 = new javax.swing.JButton();
        jToggleButton1 = new javax.swing.JToggleButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

        jPanel1.setToolTipText("Save key");

        buttonGroup1.add(jRadioButtonSmartphone);
        jRadioButtonSmartphone.setSelected(true);
        jRadioButtonSmartphone.setText("Smartphone");
        jRadioButtonSmartphone.addChangeListener(new
javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                jRadioButtonSmartphoneStateChanged(evt);
            }
        });

        buttonGroup1.add(jRadioButtonLocal);
        jRadioButtonLocal.setText("Local directory");
        jRadioButtonLocal.addChangeListener(new javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                jRadioButtonLocalStateChanged(evt);
            }
        });

        jLabel1.setText("Select the local that you want to save the key:");

        jLabelLocation.setText("Location:");

        jTextFieldDirectory.setPreferredSize(new java.awt.Dimension(10, 30));

        jButtonSelect.setText("Select");
        jButtonSelect.setPreferredSize(new java.awt.Dimension(94, 33));
        jButtonSelect.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonSelectActionPerformed(evt);
            }
        });

        jLabelIP.setText("IP Address:");

        jTextFieldPort.setEditable(false);
        jTextFieldPort.setPreferredSize(new java.awt.Dimension(84, 30));

        jLabelPort.setText("Port:");

        jTextFieldIP.setPreferredSize(new java.awt.Dimension(84, 30));
```

```
        jLabelKeyName.setText("Key name:");

        jTextFieldKeyName.setPreferredSize(new java.awt.Dimension(84, 30));

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAIL
ING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jSeparator3))
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addGap(49, 49, 49)
                                .addComponent(jRadioButtonSmartphone)
                                .addGap(39, 39, 39)
                                .addComponent(jRadioButtonLocal))
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addContainerGap()
                                .addComponent(jLabel1)))
                        .addGap(0, 0, Short.MAX_VALUE))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(30, 30, 30)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addComponent(jLabelIP)
                            .addComponent(jLabelLocation)
                            .addComponent(jLabelKeyName))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 27,
Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addComponent(jTextFieldKeyName,
javax.swing.GroupLayout.PREFERRED_SIZE, 445,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addGroup(jPanel1Layout.createSequentialGroup()
```

```
                        .addComponent(jTextFieldDirectory,
javax.swing.GroupLayout.PREFERRED_SIZE, 339,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(jButtonSelect,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jTextFieldIP,
javax.swing.GroupLayout.PREFERRED_SIZE, 240,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(18, 18, 18)
                        .addComponent(jLabelPort)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jTextFieldPort,
javax.swing.GroupLayout.PREFERRED_SIZE, 141,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                    .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                        .addComponent(jSeparator4,
javax.swing.GroupLayout.Alignment.TRAILING)
                        .addComponent(jSeparator1))))
            .addContainerGap())

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jSeparator2, javax.swing.GroupLayout.DEFAULT_SIZE, 568,
Short.MAX_VALUE)
                .addContainerGap()))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jSeparator3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
```

```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                .addComponent(jRadioButtonSmartphone)
                .addComponent(jRadioButtonLocal))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 6,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                .addComponent(jTextFieldDirectory,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jButtonSelect,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelLocation))
            .addGap(31, 31, 31)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                .addComponent(jTextFieldPort,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelPort)
                .addComponent(jTextFieldIP, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelIP))
            .addGap(25, 25, 25)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                .addComponent(jLabelKeyName)
                .addComponent(jTextFieldKeyName,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jSeparator4, javax.swing.GroupLayout.PREFERRED_SIZE, 12,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(52, Short.MAX_VALUE))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
```

```
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(150, 150, 150)
                .addComponent(jSeparator2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(183, Short.MAX_VALUE)))
        );

        jButton1.setText("Close");
        jButton1.setPreferredSize(new java.awt.Dimension(97, 33));
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jToggleButton1.setText("Save");
        jToggleButton1.setPreferredSize(new java.awt.Dimension(97, 33));
        jToggleButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jToggleButton1ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap(353, Short.MAX_VALUE)
                .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(jToggleButton1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(27, 27, 27))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(0, 0, Short.MAX_VALUE)
                    .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(0, 0, Short.MAX_VALUE)))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(311, Short.MAX_VALUE)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 34,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jToggleButton1,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addContainerGap())

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(0, 9, Short.MAX_VALUE)
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(0, 9, Short.MAX_VALUE)))
        );

        pack();
    }// </editor-fold>

    private void jRadioButtonSmartphoneStateChanged(javax.swing.event.ChangeEvent
evt) {
        jLabelIP.setEnabled(jRadioButtonSmartphone.isSelected());
        jTextFieldIP.setEnabled(jRadioButtonSmartphone.isSelected());
        jTextFieldPort.setEnabled(jRadioButtonSmartphone.isSelected());
        jLabelPort.setEnabled(jRadioButtonSmartphone.isSelected());
        jTextFieldKeyName.setEnabled(jRadioButtonSmartphone.isSelected());
        jLabelKeyName.setEnabled(jRadioButtonSmartphone.isSelected());
    }

    private void jRadioButtonLocalStateChanged(javax.swing.event.ChangeEvent evt) {
        jLabelLocation.setEnabled(jRadioButtonLocal.isSelected());
        jTextFieldDirectory.setEnabled(jRadioButtonLocal.isSelected());
        jButtonSelect.setEnabled(jRadioButtonLocal.isSelected());
    }

    private void jButtonSelectActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser folder = new JFileChooser();
        folder.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int i = folder.showSaveDialog(null);
        if (i == 1) {
            jTextFieldDirectory.setText("");
        } else {
            String path = folder.getSelectedFile().getAbsolutePath();
            jTextFieldDirectory.setText(path);
        }
    }
```

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        setVisible(false);
    }

    private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        if (GDUtils.SECRET_KEY.exists()) {
            if (jRadioButtonLocal.isSelected()) {
                String path = jTextFieldDirectory.getText();
                if ((path != null && !"".equals(path)) && !"/".equals(path)) {
                    boolean copyKey = copyKey(new File(path + "/" +
GDUtils.SECRET_KEY.getName()));
                    if (copyKey) {
                        JOptionPane.showMessageDialog(this, "The key was save with
successfully!", "Sucessful", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(this, "You need to select the location!",
"Warning", JOptionPane.WARNING_MESSAGE);
                }
            } else {
                try {
                    if (!"".equals(jTextFieldIP.getText())) {
                        if (jTextFieldKeyName.getText().length() > 5) {
                            SSLSocketClient client = new SSLSocketClient();
                            Util.createPane(this, "Wait ... Establishing communication with the
server!");
                            client.connect(jTextFieldIP.getText());
                            client.sendMessage("1");
                            client.sendMessage(jTextFieldKeyName.getText());
                            String key = KeyManager.generateSecretKeyFile();
                            client.sendMessage(key);
                            String receiveMessage = client.receiveMessage();
                            if (!"0".equals(receiveMessage) && !"".equals(receiveMessage)) {
                                JOptionPane.showMessageDialog(this, receiveMessage,
"Information", JOptionPane.INFORMATION_MESSAGE);
                                jTextFieldKeyName.setText("");
                            } else {
                                if ("0".equals(receiveMessage)) {
                                    JOptionPane.showMessageDialog(this, "Already exists a key with
the name specified!", "Information", JOptionPane.INFORMATION_MESSAGE);
                                } else {
                                    JOptionPane.showMessageDialog(this, "Key not sent", "Warning",
JOptionPane.WARNING_MESSAGE);
                                }
                            }
                            client.close();
                        } else {
                            JOptionPane.showMessageDialog(this, "The field key name must have
```

```
more than 5 characters!", "Information", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(this, "You need to insert the IP
Address!", "Information", JOptionPane.INFORMATION_MESSAGE);
                }

            } catch (ConnectException ex) {
                JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!\n" + ex.getMessage(), "Warning", JOptionPane.WARNING_MESSAGE);
            } catch (GeneralSecurityException ex) {
                JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!", "Warning", JOptionPane.WARNING_MESSAGE);
            } catch (FileNotFoundException ex) {
                JOptionPane.showMessageDialog(this, "Key not found", "Warning!",
JOptionPane.WARNING_MESSAGE);
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!", "Warning", JOptionPane.WARNING_MESSAGE);
            } catch (ClassNotFoundException ex) {
                JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!", "Warning", JOptionPane.WARNING_MESSAGE);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this, "Unable to connect to the specified
server!", "Warning", JOptionPane.WARNING_MESSAGE);
            }
        }
    } else {
        JOptionPane.showMessageDialog(this, "Key not found", "Warning",
JOptionPane.WARNING_MESSAGE);
    }
}
// Variables declaration - do not modify
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.JButton jButton1;
private javax.swing.JButton jButtonSelect;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabelIP;
private javax.swing.JLabel jLabelKeyName;
private javax.swing.JLabel jLabelLocation;
private javax.swing.JLabel jLabelPort;
private javax.swing.JPanel jPanel1;
private javax.swing.JRadioButton jRadioButtonLocal;
private javax.swing.JRadioButton jRadioButtonSmartphone;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JSeparator jSeparator2;
private javax.swing.JSeparator jSeparator3;
private javax.swing.JSeparator jSeparator4;
```

```
    private javax.swing.JTextField jTextFieldDirectory;
    private javax.swing.JTextField jTextFieldIP;
    private javax.swing.JTextField jTextFieldKeyName;
    private javax.swing.JTextField jTextFieldPort;
    private javax.swing.JToggleButton jToggleButton1;
    // End of variables declaration

    public boolean copyKey(File newfile) {
        try {
            KeyManager.storeKey(newfile);
            return true;
        } catch (FileNotFoundException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (KeyStoreException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (NoSuchAlgorithmException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (CertificateException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
        return false;
    }
}

package br.edu.utfpr.cm.antinsa.gui;

import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import br.edu.utfpr.cm.antinsa.googledrive.GoogleDriveOAuth;
import br.edu.utfpr.cm.antinsa.configuration.Config;
import br.edu.utfpr.cm.antinsa.googledrive.GoogleDriveController;
import br.edu.utfpr.cm.antinsa.security.KeyManager;
import java.io.IOException;
import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import javax.crypto.NoSuchPaddingException;
```

```
import javax.swing.JOptionPane;

public class JFramePreferences extends javax.swing.JFrame {

    private TaskBar task;
    private static GoogleDriveController driveController;
    private Thread thread;

    public JFramePreferences() {
        try {
            this.task = new TaskBar("icon.png", this, "AntiNSA");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        initComponents();
        task.instanceTask();
        getConfigGoogleDriveAccount();
        jTextFieldDefaultLocation.setText(Config.STORE_DEFAULT.getAbsolutePath());
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jTabbedPane3 = new javax.swing.JTabbedPane();
        jPanelGeneral2 = new javax.swing.JPanel();
        jLabelLocation = new javax.swing.JLabel();
        jTextFieldDefaultLocation = new javax.swing.JTextField();
        jPanelAccount = new javax.swing.JPanel();
        jTabbedPaneAccount = new javax.swing.JTabbedPane();
        jPanelGoogleDrive = new javax.swing.JPanel();
        jCheckBoxEnableGoogle = new javax.swing.JCheckBox();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jButtonAuth = new javax.swing.JButton();
        jLabelName = new javax.swing.JLabel();
        jLabelEmail = new javax.swing.JLabel();
        jButtonDeleteAccount = new javax.swing.JButton();
        jSeparator1 = new javax.swing.JSeparator();
        jSeparator5 = new javax.swing.JSeparator();
        jButtonCopyKey = new javax.swing.JButton();
        jLabel3 = new javax.swing.JLabel();
        jPanelAbout = new javax.swing.JPanel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jButtonClose = new javax.swing.JButton();
```

```
        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("AntiNSA");
        setResizable(false);
        setType(java.awt.Window.Type.UTILITY);

        jLabelLocation.setText("Default location:");

        jTextFieldDefaultLocation.setEnabled(false);

        javax.swing.GroupLayout jPanelGeneral2Layout = new
javax.swing.GroupLayout(jPanelGeneral2);
        jPanelGeneral2.setLayout(jPanelGeneral2Layout);
        jPanelGeneral2Layout.setHorizontalGroup(

jPanelGeneral2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(jPanelGeneral2Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabelLocation)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jTextFieldDefaultLocation,
javax.swing.GroupLayout.PREFERRED_SIZE, 362,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(102, Short.MAX_VALUE))
        );
        jPanelGeneral2Layout.setVerticalGroup(

jPanelGeneral2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(jPanelGeneral2Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanelGeneral2Layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.BASELINE)
                    .addComponent(jLabelLocation)
                    .addComponent(jTextFieldDefaultLocation,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(268, Short.MAX_VALUE))
        );

        jTabbedPane3.addTab("General", jPanelGeneral2);

        jPanelGoogleDrive.setMaximumSize(new java.awt.Dimension(0, 0));
        jPanelGoogleDrive.setPreferredSize(new java.awt.Dimension(612, 364));

        jCheckBoxEnableGoogle.setText("Enable Google Drive Account");
```

```
        jCheckBoxEnableGoogle.addItemListener(new java.awt.event.ItemListener() {
            public void itemStateChanged(java.awt.event.ItemEvent evt) {
                jCheckBoxEnableGoogleItemStateChanged(evt);
            }
        });

        jLabel1.setText("Name:");

        jLabel2.setText("E-mail:");

        jButtonAuth.setText("Authorization");
        jButtonAuth.setMaximumSize(new java.awt.Dimension(97, 34));
        jButtonAuth.setMinimumSize(new java.awt.Dimension(97, 34));
        jButtonAuth.setPreferredSize(new java.awt.Dimension(97, 33));
        jButtonAuth.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonAuthActionPerformed(evt);
            }
        });

        jButtonDeleteAccount.setText("Delete Account");
        jButtonDeleteAccount.setMaximumSize(new java.awt.Dimension(116, 34));
        jButtonDeleteAccount.setMinimumSize(new java.awt.Dimension(116, 34));
        jButtonDeleteAccount.setPreferredSize(new java.awt.Dimension(116, 34));
        jButtonDeleteAccount.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonDeleteAccountActionPerformed(evt);
            }
        });

        jButtonCopyKey.setText("Save key");
        jButtonCopyKey.setMaximumSize(new java.awt.Dimension(72, 34));
        jButtonCopyKey.setMinimumSize(new java.awt.Dimension(72, 34));
        jButtonCopyKey.setPreferredSize(new java.awt.Dimension(72, 33));
        jButtonCopyKey.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonCopyKeyActionPerformed(evt);
            }
        });

        jLabel3.setFont(new java.awt.Font("Cantarell", 0, 12)); // NOI18N
        jLabel3.setForeground(new java.awt.Color(255, 0, 0));
        jLabel3.setText("Click this button to save your cryptographic key.");

        javax.swing.GroupLayout jPanelGoogleDriveLayout = new
javax.swing.GroupLayout(jPanelGoogleDrive);
        jPanelGoogleDrive.setLayout(jPanelGoogleDriveLayout);
        jPanelGoogleDriveLayout.setHorizontalGroup(
```

```
jPanelGoogleDriveLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
        .addComponent(jSeparator1, javax.swing.GroupLayout.Alignment.TRAILING)
        .addComponent(jSeparator5, javax.swing.GroupLayout.Alignment.TRAILING)
        .addGroup(jPanelGoogleDriveLayout.createSequentialGroup()

.addGroup(jPanelGoogleDriveLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                .addComponent(jCheckBoxEnableGoogle)
                .addGroup(jPanelGoogleDriveLayout.createSequentialGroup()
                   .addContainerGap()

.addGroup(jPanelGoogleDriveLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                    .addGroup(jPanelGoogleDriveLayout.createSequentialGroup()
                       .addComponent(jLabel1)
                       .addGap(18, 18, 18)
                       .addComponent(jLabelName,
javax.swing.GroupLayout.PREFERRED_SIZE, 368,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(jPanelGoogleDriveLayout.createSequentialGroup()
                       .addComponent(jLabel2)
                       .addGap(18, 18, 18)
                       .addComponent(jLabelEmail,
javax.swing.GroupLayout.PREFERRED_SIZE, 366,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(jPanelGoogleDriveLayout.createSequentialGroup()
                       .addComponent(jButtonAuth,
javax.swing.GroupLayout.PREFERRED_SIZE, 110,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                       .addComponent(jButtonDeleteAccount,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))))
            .addContainerGap(123, Short.MAX_VALUE))
        .addGroup(jPanelGoogleDriveLayout.createSequentialGroup()

.addGroup(jPanelGoogleDriveLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                .addComponent(jButtonCopyKey,
javax.swing.GroupLayout.PREFERRED_SIZE, 112,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED_SIZE, 287,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(0, 0, Short.MAX_VALUE))
    );
```

```
        jPanelGoogleDriveLayout.setVerticalGroup(

jPanelGoogleDriveLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addGroup(jPanelGoogleDriveLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jCheckBoxEnableGoogle)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 8,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanelGoogleDriveLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.BASELINE)
                    .addComponent(jButtonAuth, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jButtonDeleteAccount,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanelGoogleDriveLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(jLabelName))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanelGoogleDriveLayout.createParallelGroup(javax.swing.GroupLayout.Align
ment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(jLabelEmail))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jSeparator5, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel3)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jButtonCopyKey,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(0, 44, Short.MAX_VALUE))
        );

        jTabbedPaneAccount.addTab("Google Drive", jPanelGoogleDrive);

        javax.swing.GroupLayout jPanelAccountLayout = new
javax.swing.GroupLayout(jPanelAccount);
```

```
        jPanelAccount.setLayout(jPanelAccountLayout);
        jPanelAccountLayout.setHorizontalGroup(

jPanelAccountLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanelAccountLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jTabbedPaneAccount,
javax.swing.GroupLayout.PREFERRED_SIZE, 575,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        jPanelAccountLayout.setVerticalGroup(

jPanelAccountLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanelAccountLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jTabbedPaneAccount,
javax.swing.GroupLayout.DEFAULT_SIZE, 294, Short.MAX_VALUE))
        );

        jTabbedPane3.addTab("Account", jPanelAccount);

        jLabel4.setText("AntiNSA - Version 1.0");

        jLabel5.setText("Junior Cesar de Oliveira");

        jLabel6.setText("Developed by: ");

        jLabel7.setText("Luiz Arthur Feitosa dos Santos");

        javax.swing.GroupLayout jPanelAboutLayout = new
javax.swing.GroupLayout(jPanelAbout);
        jPanelAbout.setLayout(jPanelAboutLayout);
        jPanelAboutLayout.setHorizontalGroup(

jPanelAboutLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanelAboutLayout.createSequentialGroup()

.addGroup(jPanelAboutLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
                    .addGroup(jPanelAboutLayout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jLabel4))
                    .addGroup(jPanelAboutLayout.createSequentialGroup()
                        .addGap(21, 21, 21)

.addGroup(jPanelAboutLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
```

```
EADING)
                    .addComponent(jLabel7)
                    .addComponent(jLabel5)
                    .addComponent(jLabel6))))
                .addContainerGap(372, Short.MAX_VALUE))
        );
        jPanelAboutLayout.setVerticalGroup(

jPanelAboutLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanelAboutLayout.createSequentialGroup()
                .addGap(20, 20, 20)
                .addComponent(jLabel4)
                .addGap(46, 46, 46)
                .addComponent(jLabel6)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel5)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel7)
                .addContainerGap(164, Short.MAX_VALUE))
        );

        jTabbedPane3.addTab("About", jPanelAbout);

        jButtonClose.setText("Close");
        jButtonClose.setMaximumSize(new java.awt.Dimension(97, 34));
        jButtonClose.setMinimumSize(new java.awt.Dimension(97, 34));
        jButtonClose.setPreferredSize(new java.awt.Dimension(97, 33));
        jButtonClose.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonCloseActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jTabbedPane3, javax.swing.GroupLayout.Alignment.TRAILING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(jButtonClose, javax.swing.GroupLayout.PREFERRED_SIZE,
97, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
            .addGroup(layout.createSequentialGroup()
                .addComponent(jTabbedPane3, javax.swing.GroupLayout.PREFERRED_SIZE,
348, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(jButtonClose, javax.swing.GroupLayout.DEFAULT_SIZE, 34,
Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void jButtonCloseActionPerformed(java.awt.event.ActionEvent evt) {
        this.setVisible(false);
    }

    private void jButtonCopyKeyActionPerformed(java.awt.event.ActionEvent evt) {
        if (GDUtils.SECRET_KEY.exists()) {
            saveKey();
        } else {
            JOptionPane.showMessageDialog(null, "Key not found", "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    private void jButtonDeleteAccountActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        int value = JOptionPane.showConfirmDialog(this, "Would you like to delete your
Google Drive account this computer?", "Information", JOptionPane.YES_NO_OPTION);
        if (value == JOptionPane.YES_OPTION) {
            if (value == JOptionPane.YES_OPTION) {
                deleteGoogleAccount();
                if (driveController != null) {
                    driveController.close();
                    driveController = null;
                }
                JOptionPane.showMessageDialog(null, "Your Google Drive account was deleted
this computer with success!", "Sucessful", JOptionPane.INFORMATION_MESSAGE);
            }
        }
    }

    private void jButtonAuthActionPerformed(java.awt.event.ActionEvent evt) {
        if (!GoogleDriveOAuth.isValidCredential()) {
            try {
                JOptionPane.showMessageDialog(this, "Your browser will be to open to perform
authorization!", "Information", JOptionPane.INFORMATION_MESSAGE);
                GoogleDriveOAuth.authorize();
```

```
            if (Config.STORE_DEFAULT != null) {
                Config.STORE_DEFAULT.mkdirs();
            }
            jButtonDeleteAccount.setEnabled(true);
            JOptionPane.showMessageDialog(this, "Authorization performed successfully!",
"Sucessful", JOptionPane.INFORMATION_MESSAGE);
            getKey();
            setAccountInfo();
            if (Boolean.valueOf(Config.readXMLConfig("enable-google-drive").getText()) ==
true) {
                driveController = new GoogleDriveController();
                driveController.start();
            }
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        } catch (GeneralSecurityException ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        }
    } else {
        JOptionPane.showMessageDialog(this, "The application is already authorized!",
"Information", JOptionPane.INFORMATION_MESSAGE);
    }
}

private void jCheckBoxEnableGoogleItemStateChanged(java.awt.event.ItemEvent evt) {
    setConfigGoogleDriveAccount();
    getConfigGoogleDriveAccount();
}

private void getConfigGoogleDriveAccount() {
    Boolean enable =
Boolean.valueOf(Config.readXMLConfig("enable-google-drive").getValue());
    Boolean enableAPIPersonal =
Boolean.valueOf(Config.readXMLConfig("enable-api-personal").getValue());
    jCheckBoxEnableGoogle.setSelected(enable);
    jButtonAuth.setEnabled(enable);
    if (GoogleDriveOAuth.isValidCredential()) {
        jButtonCopyKey.setEnabled(enable);
        jButtonDeleteAccount.setEnabled(enable);
    } else {
```

```
            jButtonDeleteAccount.setEnabled(false);
        }
        setAccountInfo();
    }

    private void setConfigGoogleDriveAccount() {

Config.readXMLConfig("enable-google-drive").setText(String.valueOf(jCheckBoxEnableGo
ogle.isSelected()));
        Config.saveXMLConfig();
    }

    private void setAccountInfo() {
        jLabelName.setText(Config.readXMLConfig("google-name").getText());
        jLabelEmail.setText(Config.readXMLConfig("google-email").getText());
    }

    private void deleteGoogleAccount() {
        Config.readXMLConfig("enable-google-drive").setText("false");
        Config.readXMLConfig("enable-api-personal").setText("false");
        Config.readXMLConfig("google-name").setText("");
        Config.readXMLConfig("google-email").setText("");
        Config.saveXMLConfig();
        Config.deleteDir(GDUtils.STORE_CONFIG_GOOGLE_DRIVE);
        GDUtils.SECRET_KEY.delete();
        getConfigGoogleDriveAccount();
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButtonAuth;
    private javax.swing.JButton jButtonClose;
    private javax.swing.JButton jButtonCopyKey;
    private javax.swing.JButton jButtonDeleteAccount;
    private javax.swing.JCheckBox jCheckBoxEnableGoogle;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabelEmail;
    private javax.swing.JLabel jLabelLocation;
    private javax.swing.JLabel jLabelName;
    private javax.swing.JPanel jPanelAbout;
    private javax.swing.JPanel jPanelAccount;
    private javax.swing.JPanel jPanelGeneral2;
    private javax.swing.JPanel jPanelGoogleDrive;
    private javax.swing.JSeparator jSeparator1;
```

```
    private javax.swing.JSeparator jSeparator5;
    private javax.swing.JTabbedPane jTabbedPane3;
    private javax.swing.JTabbedPane jTabbedPaneAccount;
    private javax.swing.JTextField jTextFieldDefaultLocation;
    // End of variables declaration

    private void getKey() {
        int value = JOptionPane.showConfirmDialog(this, "Would you like to create a new
key?", "Question", JOptionPane.YES_NO_OPTION);
        if (JOptionPane.YES_OPTION == value) {
            try {
                KeyManager.generateKey();
                int value1 = JOptionPane.showConfirmDialog(this, "Your key was generated and
saved in the  configuration directory of application!\nIs recommended that you save your
key in a smartphone using the application KeyManager \n or to save in another local!"
                        + "\nWould you like to save your key now?", "Information",
JOptionPane.YES_NO_OPTION);
                if (JOptionPane.YES_OPTION == value1) {
                    saveKey();
                } else if (JOptionPane.NO_OPTION == value1) {
                }
            } catch (NoSuchAlgorithmException ex) {
                JOptionPane.showMessageDialog(this, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            } catch (KeyStoreException ex) {
                JOptionPane.showMessageDialog(this, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(this, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            } catch (CertificateException ex) {
                JOptionPane.showMessageDialog(this, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            } catch (NoSuchPaddingException ex) {
                JOptionPane.showMessageDialog(this, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            } catch (InvalidKeyException ex) {
                JOptionPane.showMessageDialog(this, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            }
        } else if (JOptionPane.NO_OPTION == value) {
            receiveKey();
```

```
        }
    }

    private void saveKey() {
        JDialogSaveKey saveKey = new JDialogSaveKey(this, true);
        saveKey.setLocationRelativeTo(null);
        saveKey.setVisible(true);
    }

    private void receiveKey() {
        JDialogReceiveKey receiveKey = new JDialogReceiveKey(this, true);
        receiveKey.setLocationRelativeTo(null);
        receiveKey.setVisible(true);
        if (!GDUtils.SECRET_KEY.exists()) {
            deleteGoogleAccount();
            JOptionPane.showMessageDialog(this, "Unable to get key!\n Your account has not
been configured!", "Warning", JOptionPane.WARNING_MESSAGE);
        }
    }

    public GoogleDriveController getDriveController() {
        return driveController;
    }

    public void setDriveController(GoogleDriveController driveController) {
        this.driveController = driveController;
    }
}

package br.edu.utfpr.cm.antinsa.gui;
import java.awt.AWTException;
import java.awt.Image;
import java.awt.MenuItem;
import java.awt.PopupMenu;
import java.awt.SystemTray;
import java.awt.TrayIcon;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class TaskBar {
```

```
private static SystemTray Tray = SystemTray.getSystemTray();
private static TrayIcon TrayIconBar;
private static String IconMain;
private static JFrame frame;
private static MouseListener mlOpcoes;
private static String NameTask;
private static boolean AlterCountTray = false;

public TaskBar(String icon, JFrame frame, String nameTask) throws Exception {
    TaskBar.IconMain = icon;
    TaskBar.frame = frame;
    TaskBar.NameTask = nameTask;
}

public void instanceTask() {
    alterImageTray(IconMain);
    try {
        Tray.add(TrayIconBar);
    } catch (AWTException e) {
    }
    AlterCountTray = true;
}

public void alterImageTray(String image) {
    Image imageIcon = new ImageIcon(image).getImage();
    if (TaskBar.AlterCountTray == false) {
        TrayIconBar = new TrayIcon(imageIcon, NameTask, initPopUpMenu());
    } else {
        TrayIconBar.setImage(imageIcon);
    }
    TrayIconBar.setImageAutoSize(true);
}

public void useSequenceImage(String imageOne, String imageTwo, int time) {
    try {
        alterImageTray(imageOne);
        Thread.sleep(time);
        alterImageTray(imageTwo);
    } catch (InterruptedException ex) {
        Logger.getLogger(TaskBar.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void initEventsMouse() throws Exception {
    mlOpcoes = new MouseListener() {
        @Override
        public void mouseClicked(MouseEvent e) {
                frame.setVisible(true);
```

```
                frame.setExtendedState(JFrame.NORMAL);
            }
            @Override
            public void mousePressed(MouseEvent e) {
            }
            @Override
            public void mouseReleased(MouseEvent e) {
            }

            @Override
            public void mouseEntered(MouseEvent e) {
            }

            @Override
            public void mouseExited(MouseEvent e) {
            }
        };
        TrayIconBar.addMouseListener(mlOpcoes);
    }

    private PopupMenu initPopUpMenu() {
        PopupMenu popup = new PopupMenu();
        MenuItem miAbout = new MenuItem("About");
        miAbout.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "AntiNSA\n"
                        + "Version 1.0\n\n"
                        + "Developed by:\n Junior Cesar de
Oliveira","",JOptionPane.INFORMATION_MESSAGE);
            }
        });
        MenuItem miReturn = new MenuItem("Preferences");
        miReturn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                frame.setVisible(true);
                frame.setExtendedState(JFrame.NORMAL);
            }
        });
        MenuItem miExit = new MenuItem("Exit");
        miExit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
```

```
        popup.add(miAbout);
        popup.addSeparator();
        popup.add(miReturn);
        popup.addSeparator();
        popup.add(miExit);

        return popup;
    }

    public void displayMessageTask(String title, String message, int type) {
        String style = "";
        if (type == 0) {
            style = "NONE";
        }
        if (type == 1) {
            style = "INFO";
        }
        if (type == 2) {
            style = "ERROR";
        }
        if (type == 3) {
            style = "WARNING";
        }
        TrayIconBar.displayMessage(title, message, TrayIcon.MessageType.valueOf(style));
    }
}
```

```
package br.edu.utfpr.cm.antinsa.main;
import br.edu.utfpr.cm.antinsa.configuration.Config;
import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import br.edu.utfpr.cm.antinsa.googledrive.GoogleDriveController;
import br.edu.utfpr.cm.antinsa.googledrive.GoogleDriveOAuth;
import br.edu.utfpr.cm.antinsa.gui.JFramePreferences;
import java.io.IOException;
import java.sql.SQLException;
import javax.swing.JOptionPane;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import org.jdom2.JDOMException;

public class Main {

    public static void main(String[] args) {
        try {

javax.swing.UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName())
;
            Config.setup();
```

```
            JFramePreferences main = new JFramePreferences();
            main.setLocationRelativeTo(null);
            main.setVisible(false);
            if (Boolean.valueOf(Config.readXMLConfig("enable-google-drive").getValue()) &&
GoogleDriveOAuth.isValidCredential() && GDUtils.SECRET_KEY.exists()) {
                GoogleDriveController driveController = new GoogleDriveController();
                main.setDriveController(driveController);
                driveController.initServiceGoogleDrive();
            }
        } catch (JDOMException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        } catch (ClassNotFoundException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "SQL Error:" + ex.getMessage(),
"Warning", JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        } catch (InstantiationException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        } catch (IllegalAccessException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        } catch (UnsupportedLookAndFeelException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
JOptionPane.WARNING_MESSAGE);
            ex.printStackTrace();
        }
    }
}

package br.edu.utfpr.cm.antinsa.security;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.math.BigInteger;
import java.security.MessageDigest;
```

```
import java.security.NoSuchAlgorithmException;

public class HashGenerator {

    private static String readFile(String path) {
        String text = "";
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(path));
            String line;
            try {
                while ((line = br.readLine()) != null) {
                    text = text + line + "\n";
                }
            } catch (IOException e) {
                e.printStackTrace();
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return text;
    }

    public static String hashFile(String path) {
        String fileContent = readFile(path);
        MessageDigest md5;
        try {
            md5 = MessageDigest.getInstance("MD5");
            md5.update(fileContent.getBytes());
            return new BigInteger(1, md5.digest()).toString(16);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }
}

package br.edu.utfpr.cm.antinsa.security;
import br.edu.utfpr.cm.antinsa.configuration.Config;
import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
```

```
import java.io.ObjectOutputStream;
import java.security.InvalidKeyException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;

public class KeyManager {

    public static SecretKey loadKey() throws Exception {
        KeyStore ks = KeyStore.getInstance("JCEKS");
        ks.load(new FileInputStream(GDUtils.SECRET_KEY), "keymanager".toCharArray());
        SecretKey key = (SecretKey) ks.getKey("antinsa", "keymanager".toCharArray());
        return key;
    }

    public static boolean isValidKey(String path) {
        try {
            KeyStore ks = KeyStore.getInstance("JCEKS");
            ks.load(new FileInputStream(path), "keymanager".toCharArray());
            SecretKey key = (SecretKey) ks.getKey("antinsa", "keymanager".toCharArray());
            return true;
        } catch (Exception ex) {
            ex.printStackTrace();
            return false;
        }

    }

    public static void generateKey() throws NoSuchAlgorithmException, KeyStoreException,
IOException, CertificateException, NoSuchPaddingException, InvalidKeyException {
        KeyGenerator keygen = KeyGenerator.getInstance("AES");
        keygen.init(128);
        SecretKey key = keygen.generateKey();
        KeyStore ks = KeyStore.getInstance("JCEKS");
        ks.load(null, null);
        KeyStore.SecretKeyEntry skEntry = new KeyStore.SecretKeyEntry(key);
        ks.setEntry("antinsa", skEntry,
                new KeyStore.PasswordProtection("keymanager".toCharArray()));
        FileOutputStream fos = new FileOutputStream(new File(
                Config.STORE_CONFIG + "/antinsa.keystore"));
        ks.store(fos, "keymanager".toCharArray());
```

```
        fos.close();
    }

    public static void storeKey(File newfile) throws KeyStoreException,
FileNotFoundException, IOException, NoSuchAlgorithmException, CertificateException,
Exception {
        KeyStore ks = KeyStore.getInstance("JCEKS");
        ks.load(null, null);
        KeyStore.SecretKeyEntry skEntry = new KeyStore.SecretKeyEntry(loadKey());
        ks.setEntry("antinsa", skEntry,
                new KeyStore.PasswordProtection("keymanager".toCharArray()));
        FileOutputStream fos = new FileOutputStream(newfile);
        ks.store(fos, "keymanager".toCharArray());
        fos.close();
    }

    private static void storeKey(SecretKey key) throws KeyStoreException, IOException,
NoSuchAlgorithmException, CertificateException, UnrecoverableKeyException {
        KeyStore ks = KeyStore.getInstance("JCEKS");
        ks.load(null, null);
        KeyStore.SecretKeyEntry skEntry = new KeyStore.SecretKeyEntry(key);
        ks.setEntry("antinsa", skEntry,
                new KeyStore.PasswordProtection("keymanager".toCharArray()));
        FileOutputStream fos = new FileOutputStream(Config.STORE_CONFIG +
"/antinsa.keystore");
        ks.store(fos, "keymanager".toCharArray());
        fos.close();
    }

    public static String generateSecretKeyFile() throws FileNotFoundException,
IOException, Exception {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(loadKey());
        oos.flush();
        oos.close();
        baos.flush();
        baos.close();
        BASE64Encoder encoder = new BASE64Encoder();
        return encoder.encodeBuffer(baos.toByteArray());
    }

    public static void storeSecretKeyFile(String content) throws FileNotFoundException,
IOException, KeyStoreException, NoSuchAlgorithmException, CertificateException,
UnrecoverableKeyException, ClassNotFoundException {
        BASE64Decoder decoder = new BASE64Decoder();
        ByteArrayInputStream bais = new
ByteArrayInputStream(decoder.decodeBuffer(content));
```

```
        ObjectInputStream ois = new ObjectInputStream(bais);
        storeKey((SecretKey) ois.readObject());
        ois.close();
        bais.close();
    }
}

package br.edu.utfpr.cm.antinsa.security;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ConnectException;
import java.security.GeneralSecurityException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.SecretKey;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;

public class SSLSocketClient {

    private ObjectOutputStream oos = null;
    private ObjectInputStream ois = null;
    private SSLSocket socket = null;
    private int port = 7000;
    private String host = "";
    final String[] enabledCipherSuites = {"SSL_DH_anon_WITH_RC4_128_MD5"};

    public void connect(String host) throws ConnectException, IOException,
GeneralSecurityException {
        socket = createSSLSocket(host, port);
        oos = new ObjectOutputStream(socket.getOutputStream());
        ois = new ObjectInputStream(socket.getInputStream());
    }

    private SSLSocket createSSLSocket(String host, int porta) throws IOException,
GeneralSecurityException {
        SSLContext context = SSLContext.getDefault();
        SSLSocketFactory ssf = context.getSocketFactory();
        socket = (SSLSocket) ssf.createSocket(host, porta);
        socket.setEnabledCipherSuites(enabledCipherSuites);
        return socket;
    }

    public void sendMessage(String msg) throws IOException {
        oos.writeObject(msg);
        oos.flush();
```

```
    }

    public String receiveMessage() throws IOException, ClassNotFoundException {
        String msg = (String) ois.readObject();
        return msg;
    }
    public SecretKey receiveMessage2() {
        try {
            SecretKey s =  (SecretKey) ois.readObject();
            return s;
        } catch (IOException ex) {
            Logger.getLogger(SSLSocketClient.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(SSLSocketClient.class.getName()).log(Level.SEVERE, null, ex);
        }
        return null;
    }

    public void sendMessage2(SecretKey s) throws IOException {
        oos.writeObject(s);
        oos.flush();
    }


    public int getPort() {
        return port;
    }

    public void setPort(int port) {
        this.port = port;
    }

    public String getHost() {
        return host;
    }

    public void setHost(String host) {
        this.host = host;
    }
    public void close() throws IOException{
        socket.close();
    }
    public boolean isClosed(){
        return socket.isClosed();
    }
}

package br.edu.utfpr.cm.antinsa.security;
```

```
import br.edu.utfpr.cm.antinsa.configuration.Config;
import br.edu.utfpr.cm.antinsa.configuration.GDUtils;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

 public class SecretKeyAESCrypto {

    private Cipher cipher;
    private SecretKey key;

    public SecretKeyAESCrypto() throws Exception {
        key = KeyManager.loadKey();
        cipher = Cipher.getInstance("AES");
    }

      public File encrypt(File file) {
        try {
            cipher.init(Cipher.ENCRYPT_MODE, key);
            BufferedInputStream bufferedInputStream = new BufferedInputStream(new
FileInputStream(file));
            CipherInputStream cipherInputStream = new
CipherInputStream(bufferedInputStream, cipher);
            File tempFile = new File(GDUtils.CACHE_DIR + "/" + file.getName());
            BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(new
FileOutputStream(tempFile));
            int read = 0;
            byte[] bytes = new byte[1024];
            while ((read = cipherInputStream.read(bytes)) != -1) {
                bufferedOutputStream.write(bytes, 0, read);
            }
            bufferedOutputStream.close();
            bufferedInputStream.close();
            cipherInputStream.close();
            return tempFile;
```

```
        } catch (FileNotFoundException ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (IOException ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (Exception ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return null;
    }

    public File decrypt(File file) {
        try {
            cipher.init(Cipher.DECRYPT_MODE, key);
            BufferedInputStream bufferedInputStream = new BufferedInputStream(new
FileInputStream(file));
            CipherInputStream cipherInputStream = new
CipherInputStream(bufferedInputStream, cipher);
            File decryptedFile = new File(Config.STORE_DEFAULT.getAbsolutePath() + "/" +
file.getName());
            BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(new
FileOutputStream(decryptedFile));
            int read = 0;
            byte[] bytes = new byte[1024];
            while ((read = cipherInputStream.read(bytes)) != -1) {
                bufferedOutputStream.write(bytes, 0, read);
            }
            bufferedOutputStream.close();
            bufferedInputStream.close();
            return decryptedFile;
        } catch (FileNotFoundException ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (IOException ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (InvalidKeyException ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (Exception ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return null;
    }
```

```
    public void initCipher() {
        try {
            cipher = Cipher.getInstance("AES");
            KeyGenerator keygen = KeyGenerator.getInstance("AES");
            keygen.init(128);
            key = keygen.generateKey();
        } catch (NoSuchAlgorithmException ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (NoSuchPaddingException ex) {
            Logger.getLogger(SecretKeyAESCrypto.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }
}

package br.edu.utfpr.cm.antinsa.util;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.net.UnknownHostException;
import java.nio.channels.FileChannel;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JDialog;
import javax.swing.JOptionPane;
import javax.swing.Timer;

public class Util {

    public static String getMimeType(String absolutePath) throws IOException {
        Path path = Paths.get(absolutePath);
        return java.nio.file.Files.probeContentType(path);
```

```
}

public static String getFileName(String relativePath) throws IOException {
    Path path = Paths.get(relativePath);
    return path.getFileName().toFile().getName();
}

public static String getRelativePath(String relativePathFile) throws IOException {
    Path path = Paths.get(relativePathFile);
    return path.getFileName().toFile().getAbsolutePath();
}

public static Boolean verifyServiceConnection(String address) {
    try {
        URL url = new URL(address);
        URLConnection conn = url.openConnection();

        HttpURLConnection httpConn = (HttpURLConnection) conn;
        httpConn.connect();
        int x = httpConn.getResponseCode();
        if (x == 200) {
            return true;
        }
    } catch (UnknownHostException ex) {
        return false;
    } catch (MalformedURLException ex) {
        return false;
    } catch (IOException ex) {
        return false;
    }
    return false;
}

public static String convertInputStreamtoString(InputStream is) {
    BufferedReader br = null;
    StringBuilder sb = new StringBuilder();
    String line;
    try {
        br = new BufferedReader(new InputStreamReader(is));
        while ((line = br.readLine()) != null) {
            sb.append(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (br != null) {
            try {
                br.close();
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return sb.toString();
}

public void saveToFile(FileInputStream fis, String path) {
    try {
        FileOutputStream out = new FileOutputStream(path);
        int b;
        while ((b = fis.read()) > -1) {
            out.write(b);
        }
        fis.close();
        out.close();
    } catch (IOException ex) {
        Logger.getLogger(Util.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public static void createPane(Component comp, String msg) {
    JOptionPane pane = new JOptionPane(msg);
    final JDialog dialog = pane.createDialog(comp, "INFORMATION");
    Timer timer = new Timer(4000,
            new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            dialog.dispose();
        }
    });
    timer.setRepeats(false);
    timer.start();
    dialog.show();
    timer.stop();
    dialog.show(false);
}

public static void saveToFile(String path, String content) throws IOException {
    FileOutputStream fos = new FileOutputStream(path);
    fos.write(content.getBytes());
    fos.close();
}

public static ObjectInputStream convertStringToStream(String str) throws IOException {
    ObjectInputStream c = new ObjectInputStream(new
ByteArrayInputStream(str.getBytes()));
    return c;
```

```
    }

    public static byte[] ObjecttoBytes(Object object) {
        java.io.ByteArrayOutputStream baos = new java.io.ByteArrayOutputStream();
        try {
            java.io.ObjectOutputStream oos = new java.io.ObjectOutputStream(baos);
            oos.writeObject(object);
        } catch (java.io.IOException ioe) {
            ioe.printStackTrace();
        }
        System.out.println(baos.toString());
        return baos.toByteArray();
    }

    public static Object bytesToObject(byte[] bytes) {
        Object object = null;
        try {
            object = new java.io.ObjectInputStream(new
java.io.ByteArrayInputStream(bytes)).readObject();
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (java.lang.ClassNotFoundException ex) {
            ex.printStackTrace();
        }
        return object;
    }

    public static void copyFile(File source, File destination) throws IOException {
     if (destination.exists())
         destination.delete();

     FileChannel sourceChannel = null;
     FileChannel destinationChannel = null;

     try {
         sourceChannel = new FileInputStream(source).getChannel();
         destinationChannel = new FileOutputStream(destination).getChannel();
         sourceChannel.transferTo(0, sourceChannel.size(),
                 destinationChannel);
     } finally {
         if (sourceChannel != null && sourceChannel.isOpen())
             sourceChannel.close();
         if (destinationChannel != null && destinationChannel.isOpen())
             destinationChannel.close();
    }
}
}
```

## APÊNDICE B – CÓDIGO FONTE DO APLICATIVO KEYMANAGER

```
package br.edu.utfpr.cm.keymanager.activity;
import br.edu.utfpr.cm.keymanager.main.R;
import android.os.Bundle;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.DialogInterface;
import android.view.Menu;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class ConfigActivity extends ListActivity {

	private ListView lvOptions;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_ckm);
		lvOptions = getListView();
		lvOptions.setAdapter(new ArrayAdapter<String>(this,
				android.R.layout.simple_list_item_1, new String[] {"About" }));
		lvOptions.setOnItemClickListener(new AdapterView.OnItemClickListener() {
			public void onItemClick(AdapterView<?> parent, View view,
					int position, long id) {
					AlertDialog.Builder alert = new AlertDialog.Builder(
						ConfigActivity.this);
					alert.setTitle("About");
					alert.setMessage("KeyManager is a application for
manager the keys of the application AntiNSA. \n \n Developed by:    \n Junior Cesar de
Oliveira");
					alert.setPositiveButton("Ok",
					new DialogInterface.OnClickListener() {
	public void onClick(DialogInterface dialog, int whichButton) { } }); alert.show();
				}
		});
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		getMenuInflater().inflate(R.menu.ckm, menu);
		return true;
	}
```

```
}

package br.edu.utfpr.cm.keymanager.activity;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
import br.edu.utfpr.cm.keymanager.main.R;
import br.edu.utfpr.cm.keymanager.server.SocketServerKey;
import br.edu.utfpr.cm.keymanager.util.Config;
import br.edu.utfpr.cm.keymanager.util.Utils;

public class ServerActivity extends Activity {

	private Boolean status = false;
	private ImageButton btnPower;
	private TextView tvStatus;
	private TextView tvAddress;
	private int imgButton;
	private SocketServerKey server;
	private Thread serverThread;
	private Button btnshow;
	private Config config;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		config = new Config(getApplicationContext());
		setContentView(R.layout.activity_server);
		btnPower = (ImageButton) findViewById(R.id.imbPower);
		tvStatus = (TextView) findViewById(R.id.tvStatus);
		tvStatus.setText("Start");
		tvAddress = (TextView) findViewById(R.id.tvAddress);
		tvAddress.setText("Server stoped...");
		btnshow = (Button) findViewById(R.id.btnshow);
		btnPower.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				if (status == false
						&& Utils.isConnected(getApplicationContext())) {
					if (server == null) {
						server = new SocketServerKey(config,
								ServerActivity.this);
```

```
                        }
                        serverThread = new Thread(server);
                        serverThread.start();
                        imgButton = R.drawable.ic_power_off;
                        btnPower.setImageResource(imgButton);
                        status = true;
                        tvStatus.setText("Stop");
                        tvAddress.setText("Listening... IP "
                                        + Utils.getIPAddress(true));
                } else if (Utils.isConnected(getApplicationContext())) {
                        server.stopServer();
                        imgButton = R.drawable.ic_power_on;
                        btnPower.setImageResource(imgButton);
                        status = false;
                        tvStatus.setText("Start");
                        tvAddress.setText("Server stoped...");
                } else {
                        tvAddress
                                        .setText("The cell phone isn't connected to
NetWork...");
                }

        }
    });
    btnshow.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
                AlertDialog.Builder alert = new AlertDialog.Builder(
                                ServerActivity.this);
                alert.setTitle("Stored keys");
                String[] keys = config.getKeysName();
                String msg = "";
                if (keys.length != 0) {
                        for (String key : keys) {
                                msg += key + "\n";
                        }
                        alert.setMessage(msg);
                } else {
                        alert.setMessage("There isn't keys\n");
                }
                alert.setPositiveButton("Ok",
                                new DialogInterface.OnClickListener() {
                                        public void onClick(DialogInterface dialog,
                                                        int whichButton) {
                                        }
                                });
                alert.show();
```

```
                }
            });
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            getMenuInflater().inflate(R.menu.server, menu);
            return true;
        }

        @Override
        public void onSaveInstanceState(Bundle savedInstanceState) {
            super.onSaveInstanceState(savedInstanceState);
            savedInstanceState.putBoolean("status", status);
            savedInstanceState.putString("tvStatus", tvStatus.getText().toString());
            savedInstanceState.putString("tvAddress", tvAddress.getText()
                    .toString());
            savedInstanceState.putInt("ic_power", imgButton);
        }

        @Override
        public void onRestoreInstanceState(Bundle savedInstanceState) {
            super.onRestoreInstanceState(savedInstanceState);
            status = savedInstanceState.getBoolean("status");
            tvStatus.setText(savedInstanceState.getString("tvStatus"));
            tvAddress.setText(savedInstanceState.getString("tvAddress"));
            btnPower.setImageResource(savedInstanceState.getInt("ic_power"));
            imgButton = savedInstanceState.getInt("ic_power");
        }

}

package br.edu.utfpr.cm.keymanager.main;
import java.io.FileOutputStream;
import br.edu.utfpr.cm.keymanager.main.R;
import br.edu.utfpr.cm.keymanager.activity.ConfigActivity;
import br.edu.utfpr.cm.keymanager.activity.ServerActivity;
import android.os.Bundle;
import android.app.ActivityGroup;
import android.content.Intent;
import android.content.res.Resources;
import android.view.Menu;
import android.widget.TabHost;

public class MainActivity extends ActivityGroup {
        static TabHost tabHost;
        static int tab = 1;
        private static FileOutputStream fos;
```

```
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
		Resources res = getResources();
		tabHost = (TabHost) findViewById(R.id.tabhost);
		tabHost.setup(this.getLocalActivityManager());
		TabHost.TabSpec spec;
		Intent intent;
		intent = new Intent().setClass(this, ServerActivity.class);
		spec = tabHost.newTabSpec("0")
				.setIndicator("", res.getDrawable(R.drawable.ic_server))
				.setContent(intent);
		tabHost.addTab(spec);
		intent = new Intent().setClass(this, ConfigActivity.class);
		spec = tabHost.newTabSpec("1")
				.setIndicator("", res.getDrawable(R.drawable.ic_ckm))
				.setContent(intent);
		tabHost.addTab(spec);
		tabHost.setCurrentTab(0);
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}

	@Override
	public void onSaveInstanceState(Bundle savedInstanceState) {
		super.onSaveInstanceState(savedInstanceState);
		savedInstanceState.putInt("tab", tabHost.getCurrentTab());
	}

	@Override
	public void onRestoreInstanceState(Bundle savedInstanceState) {
		super.onRestoreInstanceState(savedInstanceState);
		tabHost.setCurrentTab(savedInstanceState.getInt("tab"));
	}
}

package br.edu.utfpr.cm.keymanager.server;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OptionalDataException;
import java.io.StreamCorruptedException;
```

```
import javax.net.ssl.SSLSocket;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.util.Log;
import br.edu.utfpr.cm.keymanager.activity.ServerActivity;
import br.edu.utfpr.cm.keymanager.util.Config;

public class Connect {

	private SSLSocket clienteSocket = null;
	private ObjectInputStream ois = null;
	private ObjectOutputStream oos = null;
	private int count = 0;
	private final String[] enabledCipherSuites =
{ "SSL_DH_anon_WITH_RC4_128_MD5" };
	private static final int SEND_KEY = 1;
	private static final int RECEIVE_KEY = 2;
	private static int typeMessage = 0;
	private static Config config;
	private static String keyName = "";
	private ServerActivity parent;

	public Connect(SSLSocket clientSocket, Config config, ServerActivity parent)
			throws Exception {
		this.config = config;
		this.parent = parent;
		clienteSocket = clientSocket;
		clienteSocket.setEnabledCipherSuites(enabledCipherSuites);
		try {
			ois = new ObjectInputStream(clienteSocket.getInputStream());
			oos = new ObjectOutputStream(clienteSocket.getOutputStream());
		} catch (StreamCorruptedException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		while (count < 3) {
			count++;
			executeRequest();
		}
		closeConnection();
	}

	public void executeRequest() throws Exception {
		String mensagemOut = "";
		String mensagemIn = receiveMessage();

		if (mensagemIn.length() == 1 && count == 1) {
```

```
                if (Integer.valueOf(mensagemIn) == SEND_KEY) {
                        typeMessage = SEND_KEY;
                } else if (Integer.valueOf(mensagemIn) == RECEIVE_KEY) {
                        typeMessage = RECEIVE_KEY;
                }
        } else if (count == 2) {
                keyName = mensagemIn;
        } else {
                if (typeMessage == SEND_KEY) {

                        if (!config.verifyKey(keyName)) {
                        config.setSharedPreferences(keyName, mensagemIn);
                        Log.v("junior", config.getSharedPreferencesKey("JUNIOR"));
                        mensagemOut = "Key sent with success!";
                        parent.runOnUiThread(new Runnable() {
                        public void run() {
                        AlertDialog.Builder alert = new AlertDialog.Builder(parent);
                                alert.setTitle("Information");
                                alert.setMessage("The key "
                                + keyName
                                + " was received from address "
                                + clienteSocket.getInetAddress() .getHostAddress()+".");
                        alert.setPositiveButton("Ok",
                        new DialogInterface.OnClickListener() {
                        public void onClick( DialogInterface dialog, int whichButton) {
                        try {
                        } catch (Exception e) {
                        e.printStackTrace();           }
                                }
                        });
                        alert.show();
                        }
                        });
                        } else {
                                mensagemOut = "0";
                        }
                } else if (typeMessage == RECEIVE_KEY) {
                        if (config.verifyKey(keyName)) {
                        mensagemOut = config.getSharedPreferencesKey(keyName);
                        parent.runOnUiThread(new Runnable() {
                        public void run() {
                        AlertDialog.Builder alert = new AlertDialog.Builder( parent);
                                alert.setTitle("Information");
                                alert.setMessage("The key "
                                + keyName
                                + " was sent to the address "
                                + clienteSocket.getInetAddress() .getHostAddress()+
".");
```

```
                                        alert.setPositiveButton("Ok",
                                new DialogInterface.OnClickListener() {
                                public void onClick( DialogInterface dialog, int whichButton) {
                                try {
                                } catch (Exception e) {
                                e.printStackTrace(); }
                                        }
                                });
                                alert.show();
                                }
                                });
                                } else {
                                        mensagemOut = "0";
                                }
                        }
                        sendMessage(mensagemOut);
                }
        }

        public void sendMessage(String mensagem) throws IOException {
                oos.writeObject(mensagem);
                oos.flush();
        }

        public String receiveMessage() throws OptionalDataException,
                        ClassNotFoundException, IOException {
                String msg = null;
                msg = ois.readObject().toString();
                return msg;
        }

        public void closeConnection() {
                try {
                        ois.close();
                        oos.close();
                        clienteSocket.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}

package br.edu.utfpr.cm.keymanager.server;
import java.io.IOException;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import br.edu.utfpr.cm.keymanager.activity.ServerActivity;
```

```
import br.edu.utfpr.cm.keymanager.util.Config;
import android.app.AlertDialog;
import android.content.DialogInterface;

public class SocketServerKey extends Thread {
	private SSLServerSocketFactory factory;
	private SSLServerSocket server;
	private SSLSocket client;
	private static final int PORT = 7000;
	private final String[] enabledCipherSuites =
{ "SSL_DH_anon_WITH_RC4_128_MD5" };
	private static int typeMessage = 0;
	private static Config config;
	private ServerActivity parent;

	public SocketServerKey(Config config, ServerActivity parent) {
		super();
		this.config = config;
		this.parent = parent;
	}

	@Override
	public void run() {
		try {
			factory = (SSLServerSocketFactory) SSLServerSocketFactory
					.getDefault();
			server = (SSLServerSocket) factory.createServerSocket(PORT);
			server.setEnabledCipherSuites(enabledCipherSuites);
			while (!isInterrupted()) {
				client = (SSLSocket) server.accept();
				parent.runOnUiThread(new Runnable() {
					public void run() {
						AlertDialog.Builder alert = new
						AlertDialog.Builder( parent);
						alert.setTitle("Information");
						alert.setMessage("The address "
						+ client.getInetAddress().getHostAddress()
						 + " wants to connect to this server"
						+ "\n Would you like authorize this connection?");
						alert.setPositiveButton("Yes",
					new DialogInterface.OnClickListener() {
	public void onClick(DialogInterface dialog, int whichButton) {
								try {
								dialog.dismiss();
						Connect c = new Connect(client,  config, parent);
								} catch (Exception e) {
								e.printStackTrace();
								}
```

```
                                                        }
                                                });
                        alert.setNegativeButton("No", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int whichButton) {
                                                        try {
                                                                client.close();
                                                        } catch (IOException e) {
                                                                e.printStackTrace();
                                                                }
                                                        }
                                                });
                                        alert.show();
                                }
                        });
                }
        } catch (IOException e) {
                e.printStackTrace();
        } catch (Exception e) {
                e.printStackTrace();
        }
    }

    public void stopServer() {
        try {
                server.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
        interrupt();
    }
}

package br.edu.utfpr.cm.keymanager.util;
import java.util.Map;
import java.util.Set;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;

public class Config {
    public static final String CONFIG = "KEY_MANAGER";
    private static Context context;
    private static SharedPreferences shared;

    public Config(Context context) {
        this.context = context;
        shared = context.getSharedPreferences(CONFIG,
Context.MODE_PRIVATE);
```

```
	}

	public void setSharedPreferences(String name, String value) {
		Editor editor = shared.edit();
		editor.putString(name, value);
		editor.commit();
	}

	public String getSharedPreferencesKey(String key) {
		return shared.getString(key, "");
	}

	public boolean verifyKey(String name) {
		return shared.contains(name);
	}

	public String[] getKeysName() {
		int count = 0;
		Map<String, ?> map = shared.getAll();
		Set<String> set = map.keySet();
		String[] keys = new String[set.size()];
		for (String key : set) {
			keys[count] = key;
			count++;
		}
		return keys;
	}
}
```

```
package br.edu.utfpr.cm.keymanager.util;
import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.http.conn.util.InetAddressUtils;
import android.content.Context;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.util.Log;

public class Utils {

	public static String bytesToHex(byte[] bytes) {
		StringBuilder sbuf = new StringBuilder();
		for (int idx = 0; idx < bytes.length; idx++) {
			int intVal = bytes[idx] & 0xff;
			if (intVal < 0x10)
				sbuf.append("0");
			sbuf.append(Integer.toHexString(intVal).toUpperCase());
```

```
		}
		return sbuf.toString();
	}

	public static byte[] getUTF8Bytes(String str) {
		try {
			return str.getBytes("UTF-8");
		} catch (Exception ex) {
			return null;
		}
	}

	public static String loadFileAsString(String filename)
			throws java.io.IOException {
		final int BUFLEN = 1024;
		BufferedInputStream is = new BufferedInputStream(new FileInputStream(
				filename), BUFLEN);
		try {
			ByteArrayOutputStream baos = new
ByteArrayOutputStream(BUFLEN);
			byte[] bytes = new byte[BUFLEN];
			boolean isUTF8 = false;
			int read, count = 0;
			while ((read = is.read(bytes)) != -1) {
				if (count == 0 && bytes[0] == (byte) 0xEF
						&& bytes[1] == (byte) 0xBB && bytes[2] == (byte)
0xBF) {
					isUTF8 = true;
					baos.write(bytes, 3, read - 3); // drop UTF8 bom marker
				} else {
					baos.write(bytes, 0, read);
				}
				count += read;
			}
			return isUTF8 ? new String(baos.toByteArray(), "UTF-8")
					: new String(baos.toByteArray());
		} finally {
			try {
				is.close();
			} catch (Exception ex) {
			}
		}
	}

		public static String getIPAddress(boolean useIPv4) {
		try {
			List<NetworkInterface> interfaces = Collections
					.list(NetworkInterface.getNetworkInterfaces());
```

```
                        for (NetworkInterface intf : interfaces) {
                                List<InetAddress> addrs = Collections.list(intf
                                                .getInetAddresses());
                                for (InetAddress addr : addrs) {
                                        if (!addr.isLoopbackAddress()) {
                                                String sAddr =
addr.getHostAddress().toUpperCase();
                                                boolean isIPv4 =
InetAddressUtils.isIPv4Address(sAddr);
                                                if (useIPv4) {
                                                        if (isIPv4)
                                                                return sAddr;
                                                } else {
                                                        if (!isIPv4) {
                                                                int delim = sAddr.indexOf('%');
                                                return delim < 0 ? sAddr :sAddr.substring(0, delim);
                                                        }
                                                }
                                        }
                                }
                        }
                } catch (Exception ex) {
                }
                return "";
        }

        public static boolean isConnected(Context context) {
                try {
                        ConnectivityManager cm = (ConnectivityManager) context
                                        .getSystemService(Context.CONNECTIVITY_SERVICE)
;
                        if (cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE)
                                        .isConnected()
                                        || cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI)
                                                        .isConnected()) {
                                return true;
                        } else {
                                return false;
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                        return false;
                }
        }

        public static String getStringFromInputStream(InputStream is) {

                BufferedReader br = null;
```

```
                StringBuilder sb = new StringBuilder();
                String line;
                try {
                        br = new BufferedReader(new InputStreamReader(is));
                        while ((line = br.readLine()) != null) {
                                sb.append(line);
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                } finally {
                        if (br != null) {
                                try {
                                        br.close();
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                }
                return sb.toString();
        }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.edu.utfpr.cm.keymanager.main"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_app"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="br.edu.utfpr.cm.keymanager.main.MainActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:configChanges="orientation" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

```
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name="br.edu.utfpr.cm.keymanager.activity.ServerActivity"
        android:label="@string/title_activity_server" >
    </activity>
    <activity
        android:name="br.edu.utfpr.cm.keymanager.activity.ConfigActivity"
        android:label="@string/title_activity_ckm" >
    </activity>
    <activity
        android:name="br.edu.utfpr.cm.keymanager.activity.KeysActivity"
        android:label="@string/title_activity_keys" >
    </activity>
    <service android:name="br.edu.utfpr.cm.keymanager.server.SocketServerKey" >
        <intent-filter>
            <action android:name="INICIAR_SERVICO" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </service>
    <activity
        android:name="br.edu.utfpr.cm.keymanager.activity.PortActivity"
        android:label="@string/title_activity_port" >
    </activity>
    <activity
        android:name="br.edu.utfpr.cm.keymanager.activity.KeyGeneratorActivity"
        android:label="@string/title_activity_key_generator" >
    </activity>
    <activity
        android:name="br.edu.utfpr.cm.keymanager.main.ConfigActivity"
        android:label="@string/title_activity_port_config" >
    </activity>
  </application>
</manifest>
```

activity_ckm.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ConfigActivity" >
    <ListView
        android:id="@+id/android:list"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:background="#696969"
        >
    </ListView>
</RelativeLayout>
```

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:android1="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#696969"
    android:paddingBottom="0dp"
    android:paddingLeft="0dp"
    android:paddingRight="0dp"
    android:paddingTop="0dp"
    android1:focusable="false"
    tools:context=".MainActivity" >
    <TabHost
        android1:id="@+id/tabhost"
        android1:layout_width="fill_parent"
        android1:layout_height="fill_parent"
        >
        <LinearLayout
            android1:layout_width="fill_parent"
            android1:layout_height="fill_parent"
            android:orientation="vertical"
            >
            <TabWidget
                android1:id="@android:id/tabs"
                android1:layout_width="fill_parent"
                android1:layout_height="wrap_content" >
            </TabWidget>
            <FrameLayout
                android1:id="@android:id/tabcontent"
                android1:layout_width="fill_parent"
                android1:layout_height="fill_parent"
                >
            </FrameLayout>
        </LinearLayout>
    </TabHost>
</RelativeLayout>
```

activity_server.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".ServerActivity" >
    <ImageButton
        android:id="@+id/imbPower"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="15dp"
        android:background="@null"
        android:src="@drawable/ic_power_on"
        android:contentDescription="@null" />
    <TextView
        android:id="@+id/tvStatus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imbPower"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="3dp"
        android:text="@null"
        android:textColor="#FFFFFF"
        />
    <TextView
        android:id="@+id/tvAddress"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/tvStatus"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="18dp"
        android:text="@null"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="#FFFFFF"
        android:textStyle="bold"/>
    <Button
        android:id="@+id/btnshow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
```

```
        android:layout_marginBottom="74dp"
        android:text="Show name of keys" />
</RelativeLayout>
```