UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Mecanismos de autenticação em serviços baseados em *Cloud*

João José Teles Gouveia

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Professor Doutor Paul Crocker
Co-orientador: Professor Doutor Simão Melo de Sousa

Covilhã, outubro de 2013

Agradecimentos

Em primeiro lugar gostaria de agradecer aos meus pais, pelo apoio que me deram durante a concretização de mais esta etapa académica, pelo enorme esforço, carinho, incentivo e confiança que depositaram nas minhas capacidades. Sem a vossa dedicação e o vosso amor incondicional seria impossível atingir este objetivo.

Da mesma forma, quero agradecer ao meu irmão por tudo isso e pelos momentos de descontração proporcionados. Ter um irmão é um presente, admiro-te muito e quero-te sempre ter ao meu lado.

Deixo um agradecimento muito especial a todos os meus restantes familiares, que de uma forma ou de outra, me apoiaram e ajudaram na conclusão desta caminhada. Sei que estão felizes por mim, e feliz eu serei graças a vocês.

Quero também agradecer à minha namorada e melhor amiga, Maria Manuel Casteleiro, porque apesar da distância que nos separava foi sempre capaz de ter uma palavra, um sorriso, a força que eu precisava para continuar a lutar pelos meus objetivos. Agradeço-te todos os dias que escolheste ficar ao meu lado enquanto trabalhava, por todos os momentos de diversão, alegria e carinho proporcionados, as palavras certas na altura certa, por todo o amor que sentes por mim. Espero algum dia conseguir retribuir na mesma proporção tudo aquilo que fizeste e fazes por mim. És sem dúvida alguma a melhor namorada do mundo, a melhor amiga, a melhor confidente. Obrigado por tudo isso e muito mais.

Agradeço profundamente ao meu orientador Professor Doutor Paul Andrew Crocker pela sua enorme dedicação, força de vontade e energia contagiante. A oportunidade que me foi dada de integrar o projeto PRICE e o seu papel como orientador foram fatores decisivos para eu considerar este como o meu melhor ano académico. As suas explicações, observações, o seu precioso tempo perdido comigo foram determinantes para o meu sucesso neste projeto. Obrigado pelas agradáveis conversas proporcionadas ao longo do ano e pela sua constante alegria e otimismo.

Igualmente, agradeço ao meu co-orientador Professor Doutor Simão Melo de Sousa por todos os minutos dispendidos para me aconselhar, todo o esforço e sentido crítico sempre presente. O que mais me contagia é a sua energia e dedicação aos projetos da sua vida. Quero realmente agradecer por todo o apoio, indispensável ao longo dos vários meses de trabalho.

Ao Ricardo Azevedo da PT Inovação o meu muito obrigado por todo o interesse demonstrado e apoio incondicional, que me foi dado ao longo do desenvolvimento da dissertação. Agradeço todas as opiniões, sugestões e reuniões onde as discussões foram extremamente benéficas para o projeto.

Não poderia deixar de agradecer ao meu amigo, colega de Mestrado e também colega no projeto PRICE João Silveira pelo espírito de entreajuda, discussões, todas as noites mal dormidas de trabalho árduo, pelos momentos de desânimo em que me apoiou e obrigou a continuar a minha caminhada e, acima de tudo, por todos os momentos de alegria, distração e boa disposição proporcionados. Obrigado por tudo isso amigo.

Não poderia faltar o meu enorme agradecimento a todos os meus amigos, pelos mais variados momentos de descontração proporcionados, por todo o apoio nesta caminhada, pelo sorriso ou abraço quando ele mais falta fazia, pela sincera amizade que mantemos e espero vir a manter para o resto da minha vida. Sem vocês a minha vida seria diferente, sem cor, com menos alegria e menos vivências. É graças a vocês também que termino mais esta fase da minha vida. O meu muito obrigado João Manuel Casteleiro, Ruben Santo, Samuel Dias, Leopoldo Ismael, Pedro Querido, Daniel Piedade, Gonçalo Félix, João Alberto Casteleiro, João Oliveira, José Varejão, Luís Ferreira, Miguel Casteleiro, Miguel Machado, Pedro Proença, Romão Vieira, Rui Santos, Ana Santos, Catarina Valério, Maria Figueiredo, Rita Mineiro e Sara Pais. Por todos os momentos passados e por todos aqueles que hão-de estar para chegar.

Resumo

Esta dissertação descreve duas arquiteturas distintas para autenticação e acesso uniforme a dados armazenados em fornecedores de armazenamento e serviços na Nuvem. A primeira arquitetura aproveita as vantagens do mecanismo de autorização *OAuth* aliado ao mecanismo de autenticação forte dos Cartões de Identidade Eletrónica Nacionais (*Eid cards*), no nosso caso o Cartão de *Eid* Português ou *Cartão de Cidadão* (CC).

Será apresentada uma comparação de mecanismos de autorização e acesso aos fornecedores de armazenamento na Nuvem, comparando os mecanismos de autorização *OAuth 1.0*, *OAuth 1.0a* e *OAuth 2.0*. Para utilizar a arquitetura proposta foi desenvolvida uma implementação que fornece acesso Web uniforme aos fornecedores de armazenamento e serviços na Nuvem mais populares, tais como a *Dropbox*, *Skydrive*, *Meo Cloud* e *Google Drive* usando o mecanismo de autenticação do Cartão de Cidadão como *Token* de acesso único. Para possibilitar o acesso uniforme a estes serviços serão descritas as diferenças entre as diferentes REST APIs pertencentes aos fornecedores considerados.

Finalmente, será apresentada a aplicação Web que permite aos utilizadores que possuam cartões de *Eid* aceder aos diferentes fornecedores de armazenamento na Nuvem considerados a partir de um ponto único de acesso.

A segunda arquitetura proposta aproveita as vantagens do mecanismo de autenticação *OpenID*. Será apresentada uma arquitetura que engloba o OpenID e o mecanismo de autenticação forte do CC. O principal objetivo deste sistema é dar ao utilizador a possibilidade de usar o seu Cartão de *Eid* (CC) para se autenticar em qualquer aplicação ou serviço Web que permita a delegação do processo de autenticação, através do protocolo OpenID.

No final, foram criadas duas aplicações Web como prova de conceito deste sistema.

Palavras-chave

Nuvem, Armazenamento, Cartões Inteligentes, Segurança, Confidencialidade, Privacidade, Autenticação, Autorização, Certificado, OAuth, OpenID, REST, JSON

Abstract

This dissertation describes two different architectures for authentication and uniform access to protected data stored on Cloud Storage Service Providers. The first architecture takes advantage of the OAuth authorization mechanism and the strong authentication mechanism of the National Electronic Identity (Eid) Cards , in our case the Portuguese Eid card or *Cartão de Cidadão* (CC).

It will be presented a comparison of authorization mechanisms and access to popular cloud storage providers, comparing the different authorization mechanisms *OAuth 1.0*, *OAuth 1.0a* and *OAuth 2.0*. Using the proposed architecture it was developed an implementation of this architecture that provides a uniform Web based access to popular Cloud Storage Service Providers such as *Dropbox*, *Skydrive*, *Meo Cloud* and *Google Drive* using the authentication mechanism of the Eid card as a unique access token. In order to provide a uniform access to these services the differences in the various REST APIs for the targeted providers will be described.

Finally the Web application that allows users that hold Eid cards a single point of access to their various cloud storage services will be presented.

The second one takes advantage of the OpenID authentication mechanism. It will be presented an architecture that includes OpenID and the strong authentication mechanism of the CC. The main objective of this system is to give the user the capability to use your Eid card (CC) to log into any application or Web service that allows the delegation of the authentication process, through the OpenID protocol.

Finally, two Web applications were created as a proof of concept of the system.

Keywords

Cloud, Storage, Smartcards, Security, Confidentiality, Privacy, Authentication, Authorization, Certificate, OAuth, OpenID, REST, JSON

Índice

Lista de Figuras

Lista de Acrónimos

API	Application Programming Interface
CA	Certification Authority
CC	Cartão de Cidadão
CRL	Certificate Revocation List
DI	Departamento de Informática
ECC	European Citizen Card
EEPROM	Electrically-Erasable Programmable Read-Only Memory
Eid	Electronic Identity
GSS-API	Generic Security Service Application Program Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IBE	Identity Based Encryption
ICAO	International Civil Aviation Organization
IdM	Identity Management System
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
IT	Instituto de Telecomunicações
JSON	Javascript Object Notation
LID	Light-Weight Identity
OAuth	Open Source Authorization Protocol
OCSP	Online Certificate State Protocol
OpenID	Open Identification Protocol
PIN	Person Identification Number
PKG	Private Key Generator
PKI	Public Key Infrastructure
PUK	PIN Unlock Key
RA	Registration Authority
REST	Representational State Transfer
SASL	Simple Authentication and Security Layer
SCEE	Sistema de Certificação Eletrónica do Estado
UBI	Universidade da Beira Interior
VA	Validation Authority
Web	World Wide Web
XML	eXtensible Markup Language
XRDS	Extensible Resource Descriptor Sequence
XRI	eXtensible Resource Identifier
Yadis	Yet another distributed identity system

Capítulo 1

Introdução

Ao longo de toda a história da Humanidade, as áreas do conhecimento têm sido alvo de constante evolução e em nenhuma destas áreas se assistiu até hoje à estagnação do conhecimento e sabedoria por ser impossível encontrar avanços necessários ao acompanhamento da evolução da história. A Informática e a Ciência de Computadores inserem-se perfeitamente neste contexto pois representam duas disciplinas nas quais as mudanças e consequentes evoluções são constantes. Deste modo, a *Cloud* representa um dos principais avanços ao nível da computação, recentemente. Juntando a este novo paradigma o crescimento exponencial da Web, onde todos os dias se assiste ao aparecimento de novas aplicações e novos serviços, é inevitável que se garanta a identidade de um utilizador, bem como a autenticidade do mesmo nos diferentes serviços a que está subscrito.

Com a expansão de aplicações e serviços disponíveis aos utilizadores surge um novo problema que necessita de rápida intervenção. A propagação de identidades digitais é um problema existente na atual conjuntura e que requere toda a atenção da comunidade científica.

Atualmente, cada utilizador possui diversos serviços Web ativos, cada qual com a sua identidade digital (normalmente desdobrada no conjunto nome de utilizador e palavra-passe), levando a um aumento das mesmas identidades com o decorrer do tempo, o que se torna claramente num problema de segurança.

O igual avanço nos últimos anos do tema *Cloud* tem alastrado este problema e é imperativo que a segurança dos dados dos utilizadores se mantenha intata apesar do aumento exponencial das suas identidades digitais. Para ir ao encontro desta necessidade surgem os sistemas de gestão de identidades. Estes sistemas têm como principais características gerir identidades individuais, a sua autenticação e autorização, papéis e privilégios para um dado serviço, ou conjunto de serviços. Estes sistemas são capazes de facilitar o acesso a dados protegidos num serviço, a terceiras entidades, sem ser necessária a partilha de informações confidenciais, como o nome de utilizador ou a palavra-passe.

Com este novo paradigma corrigem-se vários problemas ao nível de multiplicidade de identidades, autenticação e confidencialidade. O *OAuth* e *OpenID* são exemplos deste tipo de sistemas que gerem de forma particular a identidade de um utilizador. O *OAuth* surge como um protocolo focado para autorização, possibilitando a autorização para terceiros acederem a dados pessoais guardados em diferentes serviços. Já o *OpenID* recolhe toda a problemática envolvente do tema de propagação de identidades e resolve-a, surgindo como um protocolo de delegação de autenticação. Se aliado a um destes protocolos estiver a autenticação forte de cartões *Eid* podemos ter garantias de ter um sistema robusto e com uma componente importante no processo de autenticação, que é o facto de possuir um elemento físico e através deste conseguir aceder aos diferentes serviços onde o utilizador esteja registado.

1.1 Objetivo

O principal objetivo deste trabalho é criar um novo mecanismo de autenticação em ambientes *Cloud*, nomeadamente nos fornecedores de armazenamento *Cloud* mais conhecidos e utilizados. Para isso foi utilizada a autenticação forte do cartão de cidadão português. A vantagem da utilização do cartão de cidadão é clara, sendo um sistema que necessita o uso de um objeto físico (cartão), aliado a um Pin de autenticação. A utilização do cartão de cidadão em detrimento do sistema de autenticação mais utilizado (nome de utilizador e palavra-passe) eleva o grau de segurança ao nível da autenticação numa aplicação ou serviço.

Os objetivos principais previstos no plano de trabalhos são os seguintes:

- Revisão da literatura sobre serviços de autenticação de identidade em plataformas de computação *Cloud*;
- Estudo, análise e familiarização com as *APIs* dos fornecedores comuns, como o exemplo da *Dropbox*, *Skydrive* e/ou *Google Drive*;
- Estudo e análise dos protocolos de autenticação e autorização existentes;
- Especificação de um sistema de autenticação baseado no CC e proposta de alteração de um serviço de identidade para a *Cloud*;
- Instalação e configuração de um sistema de autenticação que aproveita o cartão de cidadão permitindo que um utilizador se identifique em qualquer aplicação Web.

1.2 Motivação

O presente trabalho é o resultado de uma parceria entre o grupo de investigação *RELiable and SEcure Computation* (RELEASE) da Universidade da Beira Interior e a empresa *Portugal Telecom* (PT) *Inovação* no âmbito do projeto *PRICE - Privacy, Reliability and Integrity in Cloud Environments*. Este projeto envolve vários trabalhos para além da presente dissertação e tem como objetivo a criação e disponibilização de soluções de autenticação, privacidade, confiança e integridade em plataformas *Cloud* para a possível utilização pela *PT Inovação*.

O projeto foi financiado por uma bolsa de investigação fornecida pela *PT Inovação*, através do Instituto de Telecomunicações (IT).

1.3 Contribuição

A contribuição deste projeto para as disciplinas de Informática e Ciências de Computadores é notória pois torna possível a utilização do cartão de cidadão como objeto de autenticação para qualquer serviço Web, nomeadamente para os serviços de armazenamento na *Cloud* mais comuns, como o caso da *Dropbox*, *Google Drive*, *Cloudpt* e *Skydrive*. O que potencia esta utilização global do cartão de cidadão como mecanismo de autenticação é a criação de um servidor *OpenID* proprietário que regista e inicia sessão de utilizadores que possuam este cartão. O sistema desenvolvido aplica-se a todos os serviços que aceitem a delegação de acesso a servidores

OpenID.

Sabendo que todos os fornecedores de armazenamento *Cloud* anteriormente destacados utilizam o protocolo *OAuth*, foi criada uma solução intitulada *Simploud* que permite a autorização por parte de utilizadores que possuem o cartão de cidadão aceder aos seus recursos protegidos que se encontram nesses serviços.

Para a construção desta última plataforma foi ainda criada uma *API* que integra de forma uniforme todos os pedidos *OAuth* feitos aos fornecedores de armazenamento, bem como para obtenção de informação acerca dos dados guardados nesses mesmos fornecedores.
Esta dissertação resultou ainda na publicação de um artigo científico na conferência internacional *CloudCom*[1] *'13* [GCdSA13].

1.4 Organização

A presente dissertação está organizada em vários capítulos que mostram sequencialmente o processo de investigação desenvolvido. Neste primeiro capítulo foi descrito o projeto, as motivações, os objetivos assim como a contribuição da presente dissertação. Os seguintes capítulos encontram-se organizados da seguinte forma:

- **Capítulo 2 - Estado da arte**: Neste capítulo são descritas todas as tecnologias abordadas ao longo da dissertação. Inicialmente é apresentada a problemática que originou a existência dos sistemas de gestão de identidades e são demonstrados diversos sistemas existentes, bem como protocolos de autenticação e autorização que se inserem nesta problemática. De seguida é elaborado um estudo sobre cartões inteligentes, autenticação multifatorial e especificamente sobre o cartão de cidadão português, objeto de estudo aprofundado e de trabalho nesta dissertação;
- **Capítulo 3 - Implementação**: Neste capítulo descrevem-se todas as tecnologias utilizadas, todas as ferramentas criadas e todas as componentes necessárias para a execução dos objetivos propostos. Este capítulo é dividido em dois cenários (o primeiro representa a ferramenta de acesso a dados protegidos em fornecedores externos como a *Dropbox* e o segundo cenário trata o servidor proprietário *OpenID*;
- **Capítulo 4 - Resultados**: O capítulo dos resultados pretende demonstrar, de uma forma mais visual e não conceptual, as duas implementações desenvolvidas durante este projeto. Este capítulo apresenta ainda alguns testes efetuados às duas implementações;
- **Capítulo 5 - Conclusão e Trabalho Futuro**: O último capítulo refere as conclusões do trabalho desenvolvido e descreve as tecnologias e funcionalidades que podem ser integradas de forma a complementar os sistemas apresentados nesta dissertação.

[1] http://2013.cloudcom.org/

Capítulo 2

Estado da arte

A expansão da Web como a conhecemos no presente obriga a que exista uma evolução noutros domínios da ciência de computadores e informática. Aspetos como segurança, integridade, confidencialidade, autenticidade e não-repúdio são testados, corrompidos e debatidos com o decorrer do tempo. Os desafios impostos são imensos e cada vez mais assistimos a ataques que obrigam à criação de novas arquiteturas ou protocolos, ou ainda à alteração dos já existentes. De seguida serão expostos os principais avanços na última década no campo da segurança e as principais ideias que estão a surgir para o futuro da computação e principalmente da Web, nomeadamente ao nível do aparecimento recente da infraestrutura *Cloud*.

2.1 Sistemas de gestão de identidades

Com a exponencial evolução da internet e semelhante aumento do número de serviços disponíveis para os utilizadores deparamo-nos com a propagação de identidades digitais de tal forma que no período mais recente da história da internet se tem feito um esforço enorme para arranjar soluções que ajudem a resolver os problemas relativos à explosão de identidades digitais. O igual avanço nos últimos anos do tema *Cloud* tem alastrado este problema e é imperativo que a segurança dos dados dos utilizadores se mantenha intata apesar do aumento exponencial das suas identidades digitais.

A gestão de múltiplas identidades (normalmente associadas ao par nome de utilizador e palavra-passe) não é apenas um incómodo, é também uma das principais fraquezas a nível de segurança da internet. Muitos dos serviços disponíveis possuem o seu próprio formulário de registo e obriga o cliente a registar-se para obter acesso aos seus serviços. Pior que isto é o facto adquirido de que sempre que um utilizador se regista num destes serviços vai fazê-lo usando um nome de utilizador e palavra-passe iguais ou idênticos aos já existentes, o que consiste numa má prática de segurança. Neste sentido surgiram, em 2001, os primeiros grandes esforços para corrigir esta problemática, com o aparecimento de sistemas de gestão de identidades e *standards* abertos à comunidade de autenticação e autorização.

Os sistemas de gestão de identidades, tal como o nome indica, têm como principal objetivo gerir a identidade digital de um utilizador individual, a sua autenticação, autorização, papéis e privilégios em qualquer serviço ou aplicação Web, com o intuito de melhorar a segurança, produtividade ou até mesmo tarefas repetitivas.

A evolução dos sistemas de gestão de identidades digitais acompanha o crescimento da internet, pois só desta forma é possível que um utilizador navegue com certos padrões de segurança. As principais soluções destes sistemas podem-se dividir em quatro grandes grupos:

- **Gestão de identidades**

  - ***Active Directory*** foi criada pela *Microsoft* para gerir a autenticação e autorização de todos os utilizadores e computadores num domínio de rede, ou seja, ao invés do utilizador ter uma palavra-passe para aceder ao sistema principal da empresa, uma senha para ler os seus *e-mails*, uma senha para se autenticar num computador, entre outras, com a utilização do *AD*, os utilizadores poderão ter apenas uma chave que lhe permita aceder a todos os recursos disponíveis na rede. Para este caso podemos definir uma diretoria como sendo uma base de dados que armazena as informações dos utilizadores;
  - O **Fornecimento de Contas de Utilizador** refere-se à criação, manutenção e desativação de objetos e atributos pertencentes a um dado utilizador, guardados em diversos sistemas de informação de uma empresa. Este tipo de sistemas tem como objetivo tornar mais rápido, eficiente e seguro os diferentes processos de criação, manutenção ou desativação de dados e registos de utilizador. As principais funcionalidade pelas quais se deve reger um sistema deste tipo são a sincronização de identidade (quando um utilizador altera os seus dados num dado sistema de base de dados, todas as base de dados que possuam estes dados replicados devem ser atualizadas também), o auto-fornecimento e auto-desativação (quando um utilizador se regista numa base de dados, esse utilizador deve ser automaticamente inserido em todos os outros sistemas de informação da empresa. O mesmo processo de atualização de base de dados acontece com a auto-desativação, processo em que um utilizador é eliminado do sistema), possibilidade de alteração de informação de perfil por vontade própria, permitir que utilizadores executem pedidos para aceder a determinados sistemas e aplicações, e permitir pedidos de delegação de acesso;
  - A **Sincronização de Palavra-passe** tem como principal objetivo a sincronização de uma palavra-passe entre diferentes sistemas de TI. Normalmente este processo é suportado por *software*, e a qualquer momento um utilizador pode alterar a palavra-passe, afetando sempre todos os serviços associados a este *software*. Contudo este processo é considerado menos seguro que o *Single Sign-on*, pois a descoberta da palavra-passe torna todos os serviços associados vulneráveis e inseguros.

- **Controlo de acesso**

  - Os sistemas de **Gestão de Palavra-passe** são aplicações que ajudam o utilizador a organizar as suas próprias palavras-passe e códigos PIN. Por norma estas aplicações recorrem a uma base de dados local ou a ficheiros que guardam os dados cifrados correspondentes a todas as palavras-passe do utilizador, seja para aplicações e serviços Web, seja para autenticação em outros computadores, a verdade é que este tipo de dados são comumente utilizados no dia a dia do utilizador e a memorização destes sem o apoio deste tipo de sistema seria praticamente impensável, nomeadamente para pessoas que possuam diversas palavras-passe;
  - O ***Single Sign-on*** (SSO) é uma propriedade do controlo de acesso que relaciona múltiplos sistemas de *software*, que são independentes entre si. Esta propriedade permite que um utilizador se possa autenticar e automaticamente ganhe acesso a todos os serviços relacionados com o sistema principal, sem que para isso necessite fazer autenticação entre cada um deles. Apesar deste sistema trazer os benefícios óbvios, ao tema do controlo de propagação de identidades digitais, é necessário que

este seja coadjuvado por *smartcards* ou *Tokens* OTP, pois a perda de acesso a este serviço causaria que todos os outros serviços anteriormente ligados ao SSO estariam comprometidos;

- O **Controlo de acesso baseado em papéis** tem hoje em dia um papel importante no mundo comercial. A grande maioria das empresas utiliza este sistema pois este permite que se restrinja o acesso apenas a utilizadores autorizados. Numa organização, os papéis são hoje em dia muito importantes e criados consoante as funções de trabalho dessa empresa. Não faz sentido que um empregado de escritório tenha o mesmo tipo de acesso a um sistema de informação que o diretor, por exemplo.

- Os **Serviços de diretoria** são fundamentais para o bom funcionamento dos *idM* e podem-se dividir entre repositórios de dados de identidade (para administração de atributos e contas de utilizador), replicação e sincronização de dados, virtualização de diretorias, sistemas eletrónicos de diretorias escaláveis e ainda os sistemas de próxima geração, como o CADS (*Composite Adaptive Directory Services*), cuja definição consiste num sistema de diretorias que engloba a problemática da gestão de identidades.

- Os principais **Protocolos *standard*** desenvolvidos nesta área são o *SAML*, *OpenID* e *OAuth*, possibilitando a autenticação de utilizadores (no caso dos dois primeiros) e a autorização para aceder a recursos protegidos (no caso do *OAuth*). Na seção 2.2 são descritos estes protocolos em pormenor.

2.2 Protocolos de autenticação e autorização

Como referido anteriormente, no tema dos sistemas gestores de identidades, existem diversos sub-ramos em que estes sistemas se dividem, incluindo alguns *standards* abertos disponibilizados para a comunidade e que vêm fornecer novas formas de autenticação e de autorização. Os exemplos óbvios e que sobressaem na área nestes últimos anos são os dos protocolos de autorização *OAuth* e de autenticação *OpenID*, e ainda o protocolo *SAML*, assente nas propriedades de SSO. Atualmente, estes protocolos são necessários visto que permitem que um utilizador possa partilhar recursos ou mesmo efetuar autenticação num serviço sem que para isso tenha que possuir uma conta associada ao mesmo. Deste modo, é oferecido aos programadores e utilizadores um leque muito maior de opções que podem ser inseridas e usadas nos processos de autenticação e autorização de acesso das suas aplicações.

2.2.1 Protocolo de autenticação *SAML*

O protocolo *SAML* (*Security Assertion Markup Language*) foi criado em 2001, pelo comité de segurança *OASIS*, e teve a sua última revisão em 2005, com a sua versão 2.0 [OAS05]. O *SAML* consiste numa *framework* baseada em *XML* para troca de dados de autenticação e autorização entre entidades, em particular, entre um provedor de identidade e um fornecedor de serviços. O único requisito de grande importância para esta linguagem é o *SSO*.

O *SAML* define três entidades para definir o seu mecanismo: o utilizador, o provedor de identidade (*idP*) e o fornecedor de serviço (*SP*). O primeiro passo consiste no envio de um pedido do utilizador para o *SP*, para que possa utilizar um dos seus serviços. De seguida, o *SP* faz um pedido e obtém uma afirmação de identidade por parte do *idP*. Na base desta afirmação, o *SP*

define se pretende que o utilizador aceda ao seu serviço ou não.

O protocolo *SAML* foi revisto pro duas vezes durante a sua existência, existindo as versões [OAS04] e [OAS05]. A necessidade da revisão deste protocolo prende-se com detalhes como a alteração do esquema e estrutura do protocolo, alteração das normas de assinatura digital, regras de processamento e de clarificação.

2.2.2 Protocolo de autorização *OAuth*

Como referido por Leiba [Lei12], o *OAuth* é um protocolo de autorização que se situa na categoria de sistemas de gestão de identidades, pois este permite a partilha de recursos entre serviços eletrónicos.

Este protocolo surge no ano de 2007, com a sua última revisão a ser publicada em [Ed.10] no ano de 2010, como mecanismo de autorização de acesso. O modo de operação do *OAuth* é relativamente simples e baseia-se na possibilidade de uma terceira entidade poder aceder, através do nosso consentimento, aos nossos dados protegidos num dado serviço sem que seja necessário o utilizador fornecer as suas credenciais de acesso (que normalmente se designam pelo par nome de utilizador e palavra-passe). Um caso de uso simples onde é percetível a aplicação deste protocolo é o exemplo de autorização de um serviço de impressão (consumidor), aceder a fotos privadas que se encontram noutro serviço (o fornecedor de serviço), sem que o consumidor exija ao utilizador as suas credenciais de acesso ao fornecedor de serviço. A vantagem deste mecanismo centra-se na capacidade de permitir que uma terceira entidade aceda aos dados protegidos do utilizador, sem que esta saiba quais são os dados de autenticação do utilizador no serviço que possui os dados.

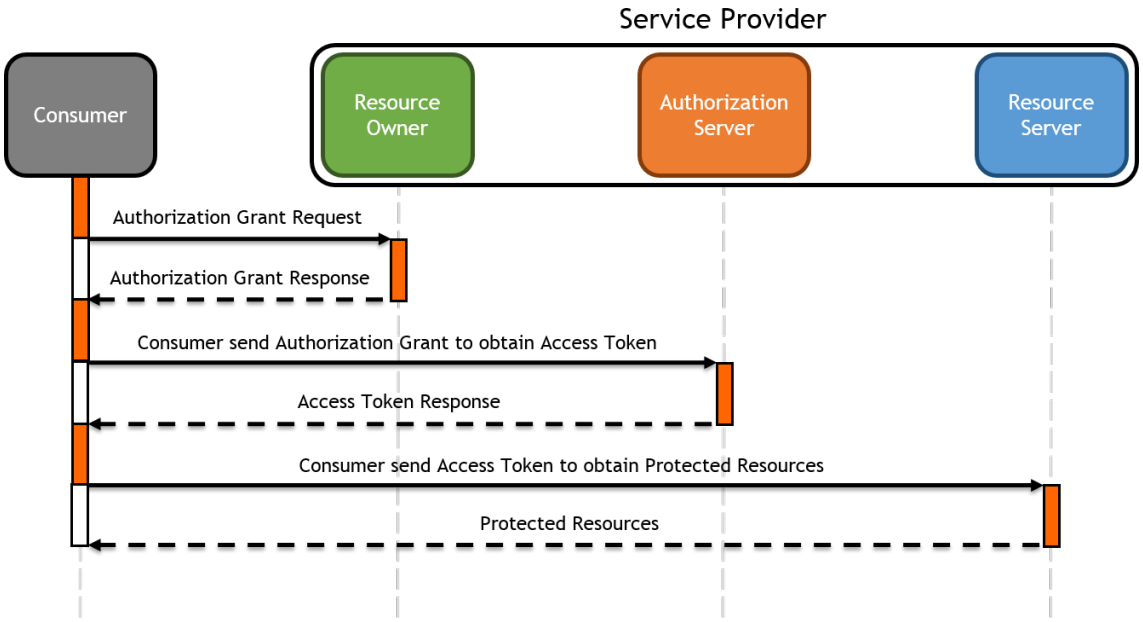

Figura 2.1: Mecanismo do protocolo de autorização *OAuth*

A Figura 2.1 é ilustrativa de como se processa todo o mecanismo deste protocolo, sendo que este pode ser dividido em três processos principais. O consumidor faz um pedido de autoriza-

ção para o proprietário dos recursos. A resposta do proprietário dos recursos deverá vir sob a forma de *token*, indicando que foi autorizado o acesso por parte do proprietário dos recursos ao consumidor. De seguida o consumidor envia um novo pedido onde inclui o *token* obtido para ser autorizado, agora pelo servidor onde estão alojados os dados confidenciais do proprietário dos recursos. Caso o proprietário permita a partilha de dados com esta terceira entidade, é enviado um novo *token*, definitivo, e até este ser revogado ou eliminado pelo proprietário dos recursos ou servidor, o consumidor terá acesso aos dados protegidos, com o consentimento prévio dado pelo proprietário.

É ainda importante referir que a cada consumidor é associado a uma chave única por parte da entidade que protege os recursos, e que todos os *tokens* gerados durante o processo de autorização do mecanismo *OAuth* são gerados com base nesta chave. O que isto quer dizer é que a esta chave ficarão associados todos os *tokens* gerados por ele, e sem ele nenhum deles funcionará. Na prática, um atacante que consiga ter acesso ao *access token* não conseguirá nada mais que isso, pois precisa da chave atribuída ao consumidor para poder assinar os pedidos feitos ao fornecedor da chave.

Para além da versão 1.0 existem hoje mais duas versões do protocolo - [et 09a] e [Ed.12].

A versão 1.0a do protocolo *OAuth* veio resolver um problema especificado em [et 09b], que se centrava num potencial ataque de fixação de sessão. O atacante podia interromper o protocolo e mais tarde fornecer o seu *url* de obtenção do *token* de autorização a outro utilizador, para de seguida receber esse mesmo *token* e poder continuar o mecanismo, neste caso com permissões de acesso aos dados do utilizador atacado. Este erro afetava de forma grave todo o mecanismo do *OAuth*, comprometendo-o, mas foi detetado sem que existissem ainda casos de uso deste potencial erro grave. Esta especificação veio corrigir este problema de forma muito simples, introduzindo um código de verificação entre o pedido de autorização e o pedido de obtenção do *token* de acesso, para que o servidor que autoriza o consumidor final saiba se é o mesmo consumidor que faz todos os pedidos.

A versão mais recente do *OAuth*, 2.0, veio introduzir novas considerações de segurança para a geração de *tokens*. As principais alterações efetuadas ao protocolo permitem agora a introdução de diferentes escopos e ainda o conceito de *Refresh Token* [M. 13]. A introdução de escopos neste protocolo foi importante pois veio possibilitar que um utilizador possa fornecer acesso a uma terceira entidade apenas a algumas partes dos seus dados. Um exemplo prático desta situação é a *Live API*, da *Microsoft*, onde o utilizador pode apenas querer partilhar os seus dados de utilizador da conta de e-mail, ou então apenas os seus dados guardados no *Skydrive*.

O conceito de *Refresh Token* foi igualmente importante pois isto permitiu que se evitassem ataques de sessão ou mesmo tentativas de obtenção dos *tokens* guardados, pois a criação deste novo conceito alterou a estrutura dos *tokens* até aqui utilizados. A principal alteração à estrutura foi a introdução de um campo de tempo de sessão, sendo que quando este campo expira o consumidor necessita iniciar o mecanismo de *Refresh Token* para obter um novo *token* válido. O mecanismo de *refresh token* é bastante mais simples. Enquanto que para se obter um *token* de acesso um consumidor apenas envia o *token* anteriormente concedido (*token* de autorização), no *Refresh Token* é necessário, para além disso, o *client_id* e *client_secret*, parâmetros que fazem parte da estrutura dos pedidos *OAuth*. Assim, este problema foi resolvido

e sempre que um *token* expira o utilizador apenas tem que usar um pedido de refrescamento para obter um novo *token* e continuar com o acesso anteriormente concedido.

2.2.3 Protocolo de autenticação OpenID

O protocolo de autenticação *OpenID* original foi desenvolvido em Maio de 2005 [Bra05] por Brad Fitzpatrick enquanto trabalhava para a empresa *Six Apart*. Foi inicialmente intitulado de projeto *Yadis* (*Yet another distributed identity system*) e só passou a ser chamado de *OpenID* quando a empresa *Six Apart* registou o domínio *openid.net* para seu uso. O *OpenID* foi rapidamente implementado no *LiveJournal*[1], um popular *website* comunitário criado por Fitzpatrick, que rapidamente ganhou a atenção da comunidade de identidades digitais. Mais tarde neste mesmo ano, iniciaram-se algumas discussões entre os utilizadores deste protocolo e programadores da empresa de *software NetMesh*, que usava o protocolo *LID* (*Light-Weight Identity*), de sua autoria. Os resultados diretos desta colaboração resultam na descoberta do protocolo *Yadis*, posteriormente intitulado *OpenID*. A entrada de programadores de *XRI* para este projeto [OAS06], contribuindo com o formato *XRDS* (*Extensible Resource Descriptor Sequence*) [OAS10], resultam na introdução deste formato no protocolo.

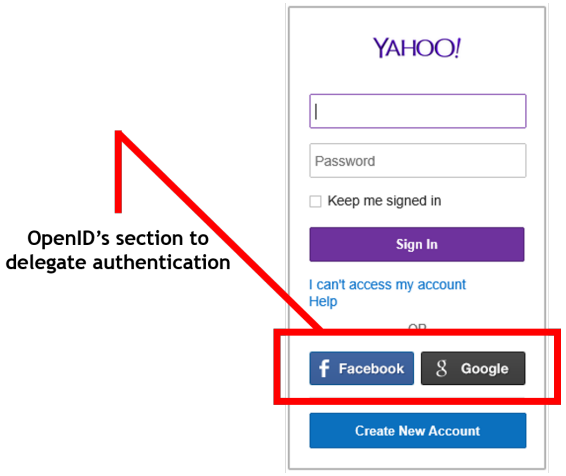

Figura 2.2: Exemplo de formulário *OpenID* na página de autenticação do *Yahoo*

O *OpenID* [RR06, Ope07] é um protocolo aberto que permite aos utilizadores autenticarem-se através de outras aplicações Web cooperantes com a aplicação que estão a aceder. Estas entidades cooperantes são conhecidas no protocolo como *Relying Party* e usam um serviço considerado como uma terceira entidade para fornecer autenticação. Com isto elimina-se a necessidade de cada serviço ou aplicação Web ter o seu próprio formulário de autenticação e consegue-se consolidar a identidade digital do utilizador. Na prática, o que este mecanismo vem potenciar é a possibilidade de um utilizador que esteja registado numa aplicação Web que suporte *OpenID* a capacidade de se registar com essa mesma identidade digital em outros serviços que permitam a delegação do processo de autenticação a um provedor *OpenID*. Ao olhar para a Figura 2.2 percebe-se as potencialidades deste mecanismo, pois para um utilizador se autenticar em toda a plataforma *Yahoo*[2] pode delegar o processo de autenticação a outros serviços, como o *Facebook* ou a *Google*.

[1] http://www.livejournal.com/
[2] https://login.yahoo.com/

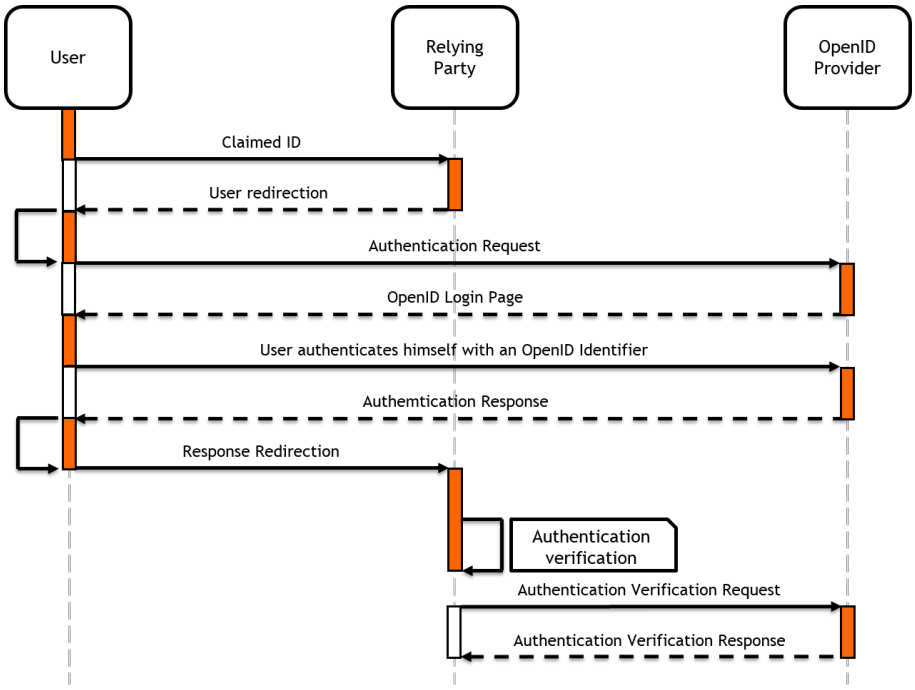

Figura 2.3: Mecanismo do protocolo de autenticação *OpenID*

A Figura 2.3 demonstra todo o mecanismo deste protocolo em pormenor. Existem cinco passos essenciais que suportam este protocolo. Primeiro o utilizador apresenta um identificador *OpenID* que pretende provar ser seu. De seguida, a entidade à qual o utilizador pretende aceder descobre qual o fornecedor *OpenID* correspondente ao identificador do utilizador e redireciona-o para este fornecedor, para que se proceda à autenticação. O utilizador autentica-se perante o seu fornecedor *OpenID* e após o sucesso da operação o fornecedor redireciona o utilizador de novo para a entidade confiante que pretende aceder. A entidade confiante recebe os dados do utilizador e estabelece um novo pedido ao fornecedor *OpenID* para validar a autenticação. O último passo deste processo passa pelo envio da resposta de validação de autenticação e, em caso do pedido ser validado, o utilizador tem finalmente acesso ao serviço que pretende usar.

Assim, caminhamos rapidamente para uma era em que uma única identidade digital servirá para que um utilizador se registe em qualquer serviço que suporte este ou outro protocolo, que siga as mesmas filosofias e mecanismos do *OpenID*.

2.3 Cartões de identificação digital

2.3.1 Práticas internacionais

Nos últimos anos, à escala global, tem-se verificado uma acentuada e progressiva criação de cartões de acesso a serviços eletrónicos, com frequentes associações a tecnologias de certificados digitais. A procura de uma maior eficiência e eficácia na prestação dos serviços e a adoção de tecnologias potenciadoras de criação de caminhos alternativos à via presencial e manual (principalmente no âmbito do acesso ao Governo Eletrónico), tem resultado numa constante preocupação de assegurar o nível de segurança adequado à prossecução das transações, pretendendo-se adequar às diferentes ameaças desenvolvidas através da criação das tecnologias

mais recentes.

Os casos mais conhecidos dentro da Europa, e que serviram de base de estudo para o Cartão de Cidadão, acontecem na **Áustria**, **Bélgica**, **Estónia**, **Finlândia** e **Suécia**. Cada um destes casos tem condicionantes específicas, pois em muitos casos a própria constituição dos países não permite certos tipos de situações relacionadas com a identificação dos cidadãos, e aquilo que se pode considerar como identificação eletrónica destes. Procede-se de seguida a uma breve análise destas caraterísticas e de algumas das condicionantes para os casos dos países mencionados, bem como de casos específicos de outros países.

No caso da **Áustria**, foi criado o conceito de cartão de cidadão que permite ao cidadão aceder a determinados serviços da Administração Pública de forma eletrónica e segura. Este caso tem a particularidade de permitir ao cidadão o uso de qualquer cartão que seja emitido por uma entidade que cumpra com os requisitos definidos para este projeto. Na Europa, esta é a situação onde existe uma maior diversidade de cartões que servem para os mesmos fins. O conceito de cartão definido pelo governo Austríaco permite que este documento digital sirva para assinatura e identificação eletrónica. O acesso ao certificado digital de autenticação encontra-se protegido por um PIN e, caso o portador do cartão falhe o código *PIN* três vezes este fica automaticamente bloqueado, sendo o desbloqueio efetuado através da introdução de um código *PUK*. Como documento físico substitui o passaporte nacional, permitindo que qualquer cidadão deste país possa viajar entre a União Europeia, Área Económica Europeia e Suíça. Os próximos passos deste conceito focam a criação de um sistema de integração de cartões *Eid* estrangeiros no sistema nacional.

Na **Bélgica**, o cartão de identidade eletrónico existe desde 2004, e surge como principal objetivo de se tornar no único instrumento para o acesso a *eServices* Públicos ou Privados. O conceito do cartão belga introduz um cartão com as mesmas funções que os cartões de identidade tradicionais e consiste no principal instrumento de identificação e autenticação para o acesso a serviços de *eGovernment*. O cartão possui ainda suporte para assinatura digital e substitui o passaporte de viagem para os países do espaço *Shengen*. Este sistema tem ainda a vantagem de possuir um registo centralizado de população atualizado.

Uma das condicionantes deste sistema é o facto da constituição belga apenas permitir que seja atribuída uma identificação aos cidadãos com idade superior a doze anos. Para contornar este problema, surgiu em 2009, o cartão *Kids ID*, que contém apenas dados comuns, como o nome, idade e contacto do cidadão. Este cartão possui um certificado de autenticação, mas não de assinatura, já que este procedimento não tem legalidade jurídica para menores de idade.

No que diz respeito à **Estónia**, o projeto para a criação de um cartão de identificação eletrónica foi iniciado em 1997 com o objetivo de fomentar o uso da identidade eletrónica a nível nacional. Este documento digital tem como funcionalidade a autenticação e assinatura digital. Este projeto foi ambicioso e a prova disso é que, em 2007, a Estónia torna-se o primeiro país a nível mundial a implementar o voto eletrónico ao nível institucional. A partir do ano de 2004, este documento substitui o documento de viagem até então utilizado, devido à entrada da Estónia na União Europeia. Os principais avanços apontados por parte deste programa indicam a interoperabilidade entre este e outros projetos homólogos de outros países, a introdução de dados biométricos para identificação eletrónica e a implementação de *WPKI* (*Wireless Public*

Key Infrastructure), arquitetura que adota os métodos *PKI* existentes para o uso em ambientes *wireless*, assegurando comércio móvel seguro.

A **Finlândia** criou o seu cartão de cidadão nacional com os objetivos de substituir o cartão de identidade nacional, o documento de viagem no espaço *Schengen* e servir como cartão de acesso a serviços eletrónicos públicos e privados que necessitem de autenticação forte. Deste modo, este conceito tem sido, desde o seu início, implementado como objeto de autenticação numa entidade bancária finlandesa (*OKO Bank*). A Finlândia é um país pioneiro na implementação do conceito de identificação eletrónica, visto que o primeiro projeto por parte das entidades governamentais surgiu em 1999, embora este não tenha sido bem sucedido. Dado que a emissão deste cartão trazia elevados custos aos cidadãos, e associando isto ao facto da licença de condução neste país permitir identificar o cidadão em praticamente todos os casos, este projeto foi dado como dispensável por parte dos cidadãos e, mais tarde, pelo estado.

O caso da **Suécia** difere de todos os outros anteriormente apresentados, pois este projeto iniciado em 2005 oficialmente, introduz um cartão de identificação eletrónica que contém um *chip*, com rádio frequência *RFID*, destinado para leitura de dados biométricos. Para além desta caraterística, este contém os dados do portador e ainda um chip utilizado para aceder a serviços de *eGovernment* de forma segura.

	Áustria	Bélgica	Estónia	Finlândia	Suécia
Chip	Sim	Sim	Sim	Sim	Sim
Certificados Digitais	Sim	Sim	Sim	Sim	Sim
Autenticação Eletrónica	Sim	Sim	Sim	Sim	Sim
Assinatura Digital	Sim	Sim	Sim	Sim	Sim
RFID	Não	Não	Não	Não	Sim
Dados biométricos	Não	Não	Não	Não	Sim

Tabela 2.1: Quadro resumo com caraterísticas de cartões internacionais

Os casos anteriormente enumerados são os casos pioneiros da Europa, sendo que nos últimos anos têm surgido mais projetos que seguem as mesmas normas, os mesmos *standards* e implementam a grande maioria dos desafios propostos anteriormente. Os casos mais flagrantes disto são os da **França**, **Alemanha**, **Holanda**, **Espanha** e **Itália**, todos eles com suporte para autenticação e assinatura digital.

O quadro 2.1, acima especificado, resume a grande maioria das caraterísticas especificadas para os projetos pioneiros europeus na área dos cartões inteligentes de identificação. Este estudo foi feito pois havia a necessidade de perceber quais as compatibilidades entre estes cartões e o principal objeto de estudo de trabalho desta dissertação, o Cartão de Cidadão Português. Todos os projetos apresentados dos diferentes países têm muitas caraterísticas em comum, seguem as mesmas normas como documentos de identificação e, em alguns casos, como documento de viagem. Podemos identificar as caraterísticas comuns através da seguinte lista:

- Todos os cartões apresentados seguem as normas estipuladas pela da *ECC* (*European Citizen Card*);
- Nos casos aplicáveis, todos aplicam as normas definidas pela *ICAO* (*International Civil Aviation Organization*) para documentos de viagem internacionais [Int08];

- Aplicação dos padrões 7501, 7810 e 7816 definidos pela *ISO* (*International Organization for Standardization*), sendo que o primeiro refere-se à criação de uma zona de leitura ótica para documentos de viagem; o segundo indica as dimensões padrão para este tipo de cartões e a terceira define, entre outras coisas, a posição do *chip* no cartão e ainda os mecanismos do *chip* e interação com estes por parte dos leitores;
- Em todos os casos de estudo é possível a identificação eletrónica, através da utilização do certificado de autenticação, e a assinatura digital de documentos;
- À exceção da Suécia, todos os restantes casos implementam uma *PKI* que serve tanto o setor público como o setor privado.

Analisando estas caraterísticas comuns a todos os cartões fabricados, e relembrando que apenas foram verificados os casos europeus, pode-se concluir que todo o trabalho efetuado no âmbito desta dissertação a nível de autenticação, através de cartões inteligentes de identificação eletrónica, pode ser facilmente utilizada por outras entidades que não apenas portadores do Cartão de Cidadão Português. Qualquer utilizador que possua um cartão de identificação eletrónica que siga as normas anteriormente apresentadas pode utilizar o sistema proposto e implementado, como os casos dos portadores de cartões *Eid* da Aústria, Bélgica, Estónia, Finlândia, Suécia e ainda os casos mais recentes da França, Alemanha, Holanda, Espanha e Itália.

2.3.2 Cartão de Cidadão Português

O Cartão de Cidadão Português começou a ser emitido em Fevereiro de 2007, como documento de cidadania, com o objetivo de substituir o bilhete de identidade, cartão de contribuinte, cartão de segurança social e cartão de utente do serviço nacional de saúde. Como documento físico, permite ao cidadão identificar-se presencialmente de forma segura. Como documento tecnológico, permite a identificação, ou autenticação, perante serviços informatizados e assinatura de documentos eletrónicos.

Na sua dimensão agregadora, junta num só documento as chaves indispensáveis ao relacionamento rápido e eficaz dos cidadãos com diferentes serviços públicos.

O Cartão de Cidadão representa um projeto de desenvolvimento tecnológico e na sua vertente digital, promove o desenvolvimento das transações eletrónicas dando-lhes a segurança da autenticação forte e da assinatura eletrónica. Os principais objetivos que lhe estão associados passam por:

- Garantia de maior segurança na identificação dos cidadãos;
- Harmonização do sistema de identificação civil dos cidadãos nacionais com os requisitos da União Europeia;
- Facilitação da vida dos cidadãos, através da agregação física de vários cartões;
- Promoção do uso de serviços eletrónicos, com recurso a meios de autenticação e assinatura digital;
- Melhoria da prestação dos serviços públicos, alinhando a modernização organizacional e tecnológica;

- Racionalização de recursos, meios e custos para o Estado, para os cidadãos e para as empresas;
- Promoção da competitividade nacional por via da reengenharia e da simplificação de processos e de procedimentos.

2.3.2.1 Caraterísticas do Cartão

O Cartão de Cidadão segue as normas internacionalmente recomendadas para que este seja um documento de identificação e um documento de viagem reconhecido oficialmente. Assim, este encontra-se alinhado pelas as orientações correntes da União Europeia, nomeadamente as do grupo de trabalho para o *ECC*, pelas normas definidas pela *ICAO* para documentos de viagem internacionais (documento 9303) e os padrões 7501, 7810 e 7816 definidos pela ISO. O formato do Cartão de Cidadão é o de um cartão *ID-1*, definido pela norma *ISO/IEC* 7810 (dimensões 85,60 x 53,98 mm) e correspondente ao formato *TD-1* de *machine readable travel documents* (MRTD) definido pela *ICAO* no Documento 9303, Parte 3, Secção III. Esta norma deve ser complementada pela norma *ISO* 7816-2, que define a posição e funcionamento do *chip* embebido no cartão.

Em função dos objetivos de utilização, das aplicações previstas e das soluções perspectivadas, o chip do Cartão de Cidadão tem as seguintes características:

- É um *chip Java Card*, multi-aplicação;
- Suporta a versão mais recente da plataforma *Java Card* e o uso de *logical channels*;
- Possui uma capacidade de memória *EEPROM* (ou equivalente) mínima de 64KB;
- Suporta múltiplos *PINs*. Os *PINs* estão em conformidade com as normas apresentadas anteriormente;
- Suporta mecanismos de bloqueio em caso de erro na introdução do *PIN* após *N* tentativas e respetivo desbloqueio por meio da introdução de *PUK* e chave administrativa de acesso ao cartão;
- Suporta mecanismos de geração de novo *PIN* mediante a introdução do *PUK* do cidadão;
- Possui um motor criptográfico interno que suporta:
  - Assinatura e verificação *RSA* de 1024 bits;
  - Assinatura digital qualificada;
  - *DES* e *TDES* (*Data Encryption Standard* e *Triple Data Encryption Standard* respetivamente);
  - *MD5*, *SHA-1* e *SHA-256*, no mínimo;
  - *MAC* (*Message Authentication Code*);
  - *PKCS#1* (*RSA Cryptography Standard*) e *PKCS#15* (*Cryptographic Token Information Format Standard*);
  - é compatível com leitores de cartões na norma *EMV-CAP*, para funcionamento de autenticação multi-canal baseada em *One Time Password*;
  - Resistente a ataques conhecidos como *"Hardware attack"*, *"Timing attack"*, *"Simple power analysis"* e *"Differential power analysis"*.

A nível físico o cartão de cidadão tem um aspeto idêntico ao apresentado na Figura 2.4, contém diversas informações acerca do cidadão, como o nome, data de nascimento, nacionalidade, fotografia e os diversos números de identificação representativos dos diferentes serviços públicos do estado, cujo cartão vem substituir.

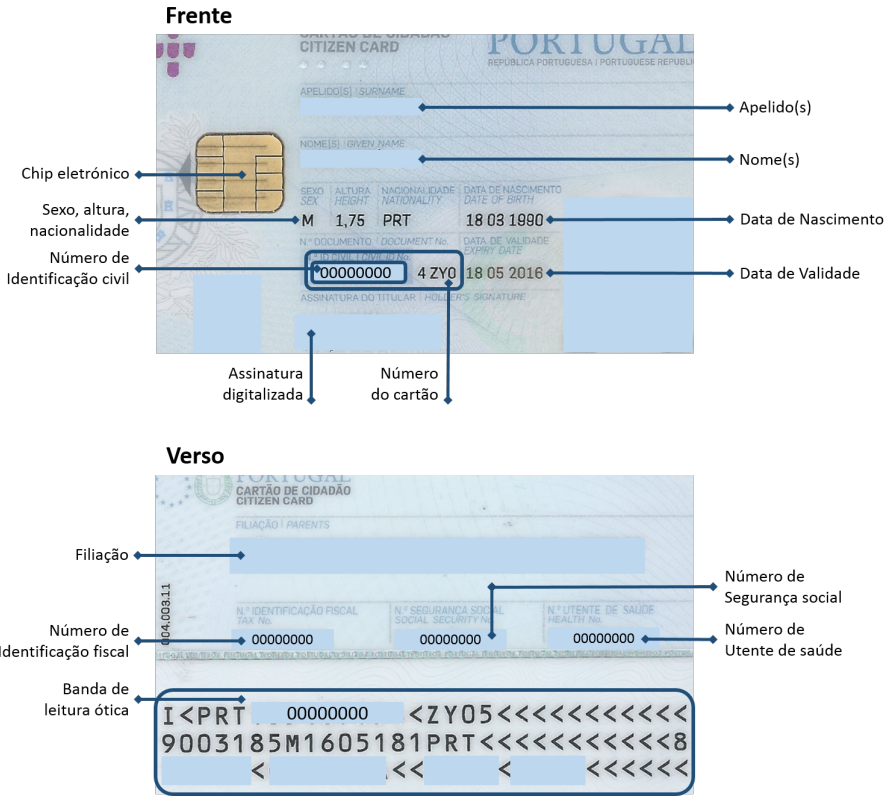

Figura 2.4: Identificação dos dados visíveis no Cartão de Cidadão

O cartão foi lançado, como parte de um plano de desenvolvimento tecnológico, com os objetivos de juntar num só documento, os documentos necessários em vários serviços públicos e promover a identificação eletrónica através do mecanismo de autenticação forte e da assinatura digital e a interação com diferentes serviços públicos e privados. As principais funcionalidades pensadas para este *smartcard* aquando do seu lançamento são:

- Guardar informação privada que o titular pode utilizar mas não conhecer nem partilhar. Dentro deste campo podemos identificar três diferentes chaves criptográficas:
  - Chave simétrica de autenticação do titular;
  - Chave privada de um par de chaves assimétricas RSA para autenticação do titular;
  - Chave privada de um par de chaves assimétricas RSA para assinatura digital;
- Guardar informação pessoal para validação interna de identidade do titular. Esta validação consiste num mecanismo de *Match-on-Card*, que permite comparar os dados biométricos guardados dentro do *smartcard*, neste caso específico a comparação entre a impressão digital do titular guardada no cartão e uma impressão digital comunicada ao mesmo, para validação de identificação;
- Guardar informação reservada. Esta informação refere-se unicamente à morada do titular do cartão e pode ser disponibilizada a quem desejar ou a quem tiver autorização para

obter esta informação, sendo necessário para isso que o titular do cartão introduza um *PIN* de desbloqueio desta informação;

- Guardar informação pública. Esta informação de conhecimento público, como referido anteriormente, consiste na fotografia digital do titular do cartão, nos certificados de chave pública de autenticação e assinatura digital e ainda de toda a informação do titular visível fisicamente no cartão;
- Efetuar operações criptográficas, através da utilização das chaves privadas fornecidas pelo cartão.

O cartão possui ainda três códigos *PIN*, cada um constituído por quatro algarismos, para efetuar três tipos de operações:

- Autorização de indicação de morada;
- Autenticação eletrónica do titular do cartão;
- Produção de assinatura digital por parte do titular do cartão.

A camada adicional de segurança que se consegue com a introdução destes códigos PIN no cartão impedem que, em caso de extravio ou perda do cartão, outra entidade que esteja na posse deste documento físico não possa aceder a dados confidenciais do titular do cartão, nem fazer uso das funcionalidades deste *smartcard*.

2.3.2.2 Cadeia de certificados e Certificação

No campo da certificação existem muitos passos até que um certificado esteja pronto para executar a tarefa para a qual foi criado, como por exemplo a autenticação digital. Os certificados do Cartão de Cidadão não são alheios a todo este processo de certificação. Mas para se perceber todo o processo de certificação de um certificado é necessário começar pelo início de todo o processo, sendo para isso necessário falar de *PKI*, ou caso a inexistência desta, explicar o que é uma *CA* (*Certification Authority*).

Como definição de *PKI* pode-se dizer que é o conjunto de todo o *hardware*, *software*, pessoas, políticas e procedimentos para criar, gerir, distribuir, usar ou revogar certificados digitais. Normalmente uma *PKI* designa-se por um órgão, público ou privado, que gere uma estrutura de emissão de chaves públicas, baseando-se sempre no princípio de terceira parte confiável, estabelecendo a ponte de credibilidade e confiança em transações entre partes que utilizam certificados digitais. Uma *CA* é a entidade certificadora da infra-estrutura de chaves públicas e o seu principal objetivo é gerir a ligação de chaves públicas a outras entidades, sendo que cada entidade deve ser considerada única no domínio da *CA* para que se possa estabelecer a ligação entre esta e uma chave pública gerada pela *CA*. Numa infraestrutura *PKI* podem-se ainda considerar as entidades *VA* e *RA* (*Validation Authority* e *Registration Authority*, respetivamente). Enquanto que a *VA* exerce o papel de registo de um utilizador na infraestrutura, a segunda atribui-lhe uma chave pública, concluindo o processo de ligação entre os dois.

O estado português, a par de vários exemplos descritos em na Secção 2.3.1, criou então a sua própria estrutura de gestão de certificados (X.509), conhecida por *SCEE*, ou *Sistema de Certificação Eletrónica do Estado*. A arquitetura da *SCEE* constitui assim uma hierarquia de confiança,

que garante a segurança eletrónica do Estado. Para o efeito a *SCEE* compreende uma Entidade Gestora de Políticas de Certificação que aprova a integração de entidades certificadoras na *SCEE* pronunciando-se igualmente sobre práticas e políticas de certificação, uma Entidade Certificadora Eletrónica Raiz, que constitui o primeiro nível da cadeia hierárquica de certificação, e as várias Entidades Certificadoras do Estado a esta subordinadas.

Podemos assim dizer que a *PKI* do estado funciona então como intermediária entre os portadores do CC e as entidades confiáveis de raiz. A entidade de raiz que se encontra no topo da cadeia de certificação dos certificados do Cartão de Cidadão é a *GTE Cyber Trust Global Root*, uma entidade confiável a nível mundial que emitiu o certificado *ECRaizEstado* que, consequentemente, herdou o nível de fiabilidade da primeira. Posto isto, a *PKI* do estado possuidora do seu próprio certificado emitido por uma entidade confiável por si só, é capaz de criar, gerir e revogar os seus próprios certificados. Ao olhar para a Figura 2.5, podemos analisar esta situação e como está estruturada toda a cadeia de certificados do Cartão de Cidadão.

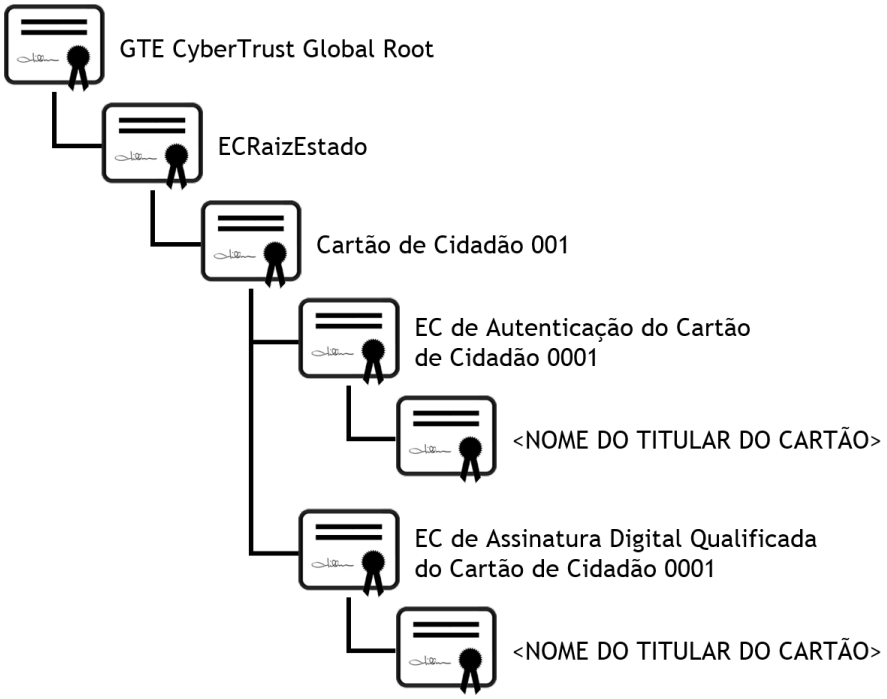

Figura 2.5: Cadeia dos certificados de chave pública de autenticação eletrónica e assinatura digital do Cartão de Cidadão Português

Para além do certificado de autenticação identificado na Figura 2.5 (*EC de Autenticação do Cartão de Cidadão 0001*) existem ainda mais sete certificados, criados para o mesmo fim. A razão pela qual a *SCEE* estabeleceu este tipo de arquitetura de certificados é que caso, por algum motivo, exista uma falha no sistema ou um possível ataque contra este certificado, apenas os certificados inferiores nesta cadeia serão afetados. Na prática esta situação faria com que apenas os cidadãos que tivessem este certificado na cadeia de certificação fossem obrigados a fazer um novo cartão, e não toda a população que possua o Cartão de Cidadão.

Outro aspeto importante a considerar na certificação é existirem processos de verificação e validação de identidade. É estritamente necessário que, em todo este sistema de certificação, a Entidade Certificadora possua mecanismos que provem a veracidade, autenticidade e validade dos certificados apresentados a esta, isto para que uma terceira entidade que queira verificar,

autenticar e validar, digitalmente, uma entidade possuidora de certificados possa enviar um pedido para a CA desse mesmo certificado e esta lhe possa responder afirmativa ou negativamente, consoante o estado do certificado do cidadão.

Os mecanismos de validação de certificados mais conhecidos, e suportados pela *SCEE*, são os métodos *OCSP* (*Online Certificate State Protocol*) [MAM+99] e *CRL* (*Certificate Revocation List*) [CSF+08].

O primeiro mecanismo aparece em substituição do segundo e na correção de alguns problemas comuns às *CRLs*. As *CRLs* são ficheiros atualizados e mantidos pela *CA*, numa média global de vinte e quatro em vinte e quatro horas. É fornecido um caminho para que qualquer entidade que queira comprovar a validade de um certificado gerado pela *CA* possa descarregar esta lista e proceder a essa verificação. Um dos problemas das *CRLs* está associado a esta troca de informação pela rede inclusive o processamento dos dados no lado do cliente. O mecanismo *OCSP* vem corrigir esse problema, pois é a *CA* que processa o pedido do cliente e fornece uma resposta booleana relativa ao estado do certificado apresentado. Para além disto, as *CRLs* sofriam de outro problema: o tempos de atualização das listas. Como exemplo, caso uma lista seja apenas atualizada de hora a hora e no espaço de uma hora um certificado tenha sido revogado, qualquer pedido de certificação desse certificado será dado como válido, quando na realidade não o é.

O uso do mecanismo *CRL* pode-se considerar como uma vantagem, principalmente para ambientes que não tenham uma ligação de rede fluída e estável, pois este não exige uma ligação constante à rede para obter o estado dos certificados. No nosso caso, optou-se pela utilização deste mecanismo dado que se verifica esta situação.

No caso específico do Cartão de Cidadão a *CA* tem permissões para emitir, validar, suspender e revogar um certificado. Noutros casos uma *CA* poderia renovar certificados, mas essa prática não é suportada pela *SCEE*.

2.3.2.3 Autenticação eletrónica

Como visto anteriormente, a autenticação de um utilizador dentro de um serviço eletrónico pode ser efetuada de diversas formas. A utilização do método nome de utilizador e palavra-passe é, atualmente, a forma mais utilizada para autenticação, mas o avanço da tecnologia requere outros métodos, outras tecnologias. A introdução de certificados digitais como método de autenticação eletrónica veio permitir adicionar uma camada de segurança nos mecanismos de autenticação. A entrada dos certificados na segurança eletrónica veio permitir que, não só possibilite autenticação de um utilizador, como a assinatura de documentos, excertos de texto, *mails*, entre outros, e ainda a comunicação segura entre a aplicação e o cliente, por via *SSL*.

Existem hoje cada vez mais fatores de autenticação, e podemos classificá-los em três grupos distintos:

- Identificação do utilizador. Podem-se incluir neste grupo fatores como impressão digital, retina do olho, sequência de ADN, reconhecimento de voz e assinatura ou qualquer outro dado biométrico;
- O que o utilizador tem, seja um cartão de identificação, *Token* de segurança, *Token* de

software ou telefone;

- O que o utilizador conhece, como a palavra-passe, frase de segurança ou *PIN*.

Aliando a estes fatores um objeto físico que agrega certificados digitais e mecanismos de autenticação temos em posse um documento suportado por fatores muito fortes de segurança, que engloba todos os fatores de autenticação.

A autenticação com CC pode ser realizada de duas formas, seja por *EMV-CAP* (*Europay, Mastercard and Visa Chip Authentication Program*) ou através do par de chaves assimétricas de autenticação. Com o primeiro método, o utilizar consegue gerar uma *OTP* (*One-time Password*) inserindo o cartão num leitor pessoal e introduzindo o *PIN* de autenticação. A *OTP* obtida pode ser enviada a uma entidade que a consiga verificar e assim autenticar o titular. Usando o segundo método, cada vez que o titular pretenda usar a sua chave privada do par de chaves assimétricas de autenticação, o *PIN* tem que ser enviado para o *smartcard*. O *smartcard* possui um certificado X.509 com a chave pública de autenticação, que pode ser comunicado, posteriormente verificado e a chave privada de autenticação validada.

2.3.2.4 Middleware

O *middleware* desenvolvido pelo estado português permite a interação entre o sistema operativo do *smartcard* do CC e o sistema operativo do computador do cliente e integra três interfaces diferentes, visto que o principal objetivo de desenvolvimento do *middleware* é o funcionamento multi-sistema operativo. São elas:

- *Crypto API*/*CSP*;
- *PKCS#11*;
- *eID lib* (biblioteca de desenvolvimento).

A primeira apenas se encontra disponível para sistemas *Windows*, pois a comunicação do *CSP* (proprietária do *middleware*) com as aplicações é mediada pela *Crypto API*, enquanto que as outras duas interfaces se encontram disponíveis para *Windows*, *Linux* e *Mac OS X*. As duas primeiras interfaces são *standards* para operações criptográficas, enquanto que a última é principalmente direcionada para operações não criptográficas, isto é, para manipulação das informações básicas como nome, números de identificação, fotografia, entre outras.

2.4 REST APIs

Recentemente, a arquitetura *REST* (*REpresentational State Transfer*) foi proposta como alternativa aos serviços Web existentes [Fie00]. Nesta era, marcada pelo aparecimento da *Cloud* e da expansão da internet, muitos serviços eletrónicos têm adotado esta arquitetura, principalmente devido ao aumento do número de dispositivos móveis e de todos os serviços feitos para eles.

Esta arquitetura possui diversas vantagens em relação às até então existentes. Para além de ser uniforme e atribuir a cada recurso, ou informação, o seu próprio identificador, este é ainda um sistema fácil de construir, escalável e que comunica baseado em "pedidos/resposta", assente no

protocolo *HTTP*, aproveitando os pedidos do protocolo *HTTP* como *GET* e *POST*. Em verdade, todos esses pedidos são perfeitamente mapeados para as necessidades de aplicações com sistemas de informação por trás, podendo equivaler os métodos *POST*, *GET*, *PUT*, *DELETE* a Criar, Ler, Atualizar e Remover.

As respostas enviadas para o utilizador podem seguir vários formatos, como *XML* ou *HTML*, mas o mais comum nos dias de hoje é o formato *JSON*. Este formato é nativo da linguagem *Javascript* e é baseado na notação *"atributo : valor"* onde o *atributo* representa o nome com o qual identificamos uma propriedade e o *valor* o atual valor dessa propriedade. Apesar de nascer dentro e para esta linguagem, este formato de resposta ganhou muita força, principalmente devido ao fato do *JSON* poder representar confortavelmente estruturas diferentes.

Devido a esta conjetura, alguns dos serviços de armazenamento na nuvem possuem já as suas próprias *REST APIs*, como é o caso do *Dropbox*, *Skydrive*, *Google Drive*, *MEO Cloud*, que foram objeto de estudo detalhado no âmbito desta dissertação. Todos estes fornecedores de armazenamento implementaram as suas *APIs* proprietárias, os seus próprios objetos e também o formatos *JSON* mais convenientes para cada objeto criado. Este aspeto vem dificultar a tarefa de construir uma *API* uniforme para interação com todos eles.

2.5 Trabalhos relacionados

Conforme os objetivos da dissertação não só é importante a análise de sistemas que implementam *REST* para se perceber como funcionam estas arquiteturas, como é necessário avaliar as capacidades dos protocolos de autenticação e autorização aliadas com *Eid smartcards* que possuam mecanismos de autenticação forte. Muitos trabalhos já foram desenvolvidos na área e têm por base a utilização de muitas destas componentes.

Pascal Urien é o exemplo claro disso mesmo e em [Uri10] descreve um esquema baseado no protocolo de autenticação *OpenID* e um sistema de autenticação baseado em *smartcards*. O esquema especificado neste artigo é relativamente simples. A autenticação do utilizador é feita através de sessão *SSL*, aberta automaticamente pelo *smartcard*, aproveitando a autenticação forte oferecida por este dispositivo, isto é, ambos os lados lidam com certificados X.509 e chave privada *RSA*.

O primeiro passo do mecanismo do sistema passa por ligar um *smartcard* a um dispositivo *USB*, com memória flash. Consequentemente este dispositivo é conectado ao computador e é estabelecida automaticamente a ligação entre o *smartcard* e o computador do utilizador. Ao iniciar uma sessão *HTTP* com o consumidor é mostrado um pequeno formulário para que o utilizador introduza o seu *OPID* (OP indentifier). Mais tarde, e conforme o utilizador necessite de se autenticar, será iniciado o processo apelidado de *Discovery*, que representa o passo em que a Entidade Confiante vai à procura do fornecedor *OpenID* correspondente ao *OPID* do utilizador. Posto este passo procede-se à validação da autenticação do utilizador. Enquanto que nas implementações clássicas de provedores a autenticação do utilizador é feita através da introdução da palavra-passe, no sistema de Urien a autenticação do utilizador é feita através de uma sessão *SSL* com autenticação mútua. Todo o restante processo de autenticação através do protocolo *OpenID* é seguido sem alterações.

Mais recentemente, Urien, implementa um novo sistema [Uri11], muito idêntico ao apresentado anteriormente, onde a principal diferença incide na introdução do mundo móvel e da possibilidade de utilização de *smartcards* das operadoras telefónicas, desde que estes possuam as funcionalidades de segurança necessárias.

Outro trabalho que envolveu o protocolo *OpenID* foi a criação de um padrão publicado em *RFC* [LTHM12] onde a principal contribuição deste foi a especificação de um mecanismo para as *frameworks SASL* e *GSS-API* que permite a integração dos provedores de autenticação *OpenID* em aplicações que usem estas *frameworks*.

Os sistemas de Urien incidem dentro do segundo cenário proposto para esta dissertação, pois os objetivos do sistema proposto são fundamentalmente iguais, com a diferença de que esta implementação funciona para cartões de identificação eletrónica nacionais, como o caso do Cartão de Cidadão Português. Apesar desta semelhança entre os sistemas de Urien e o nosso, explicado mais detalhadamente à frente, a primeira solução proposta neste trabalho está relativamente dispersa dos fundamentos destas. Enquanto que Urien utiliza um servidor *OpenID* baseado em autenticação através de certificados, com sessão *SSL*, a primeira arquitetura implementada neste trabalho utiliza o protocolo *OAuth* para autorização de uma terceira entidade, no nosso caso o CC, a aceder a recursos protegidos.

Existem igualmente diversos trabalhos na área da autorização utilizando o protocolo *OAuth*, como [AS11a, AS11b], onde Al-Sinani descreve uma arquitetura que integra sistemas de cartões de informação, como o *CardSpace* (projeto atualmente descontinuado) ou o *Higgins*, e protocolos de autenticação e autorização, como o *OpenID* e o *OAuth*.

Como já foi referido várias vezes, nesta dissertação foi considerada a utilização do Cartão de Cidadão que permite autenticação forte em serviços públicos e privados. Os cartões nacionais de *Eid* como o CC nasceram também como *token* de autenticação para serviços governamentais, como é o caso em Portugal onde o Cartão de Cidadão é atualmente utilizado para efetuar o processo de autenticação no Portal das Finanças Nacional. Mas outras implementações e estudos já surgiram para o CC, estando entre elas um sistema de autenticação *e-Health* para profissionais de saúde [GCZ07], e ainda um sistema de gestão de identidades federado [PTP10].

O *REST* é uma arquitetura cheia de potencialidade e que trouxe grandes vantagens aos serviços *Web* e a todas as aplicações na internet. Como visto na secção 2.4 esta é uma arquitetura perfeitamente escalável, que possibilita a atribuição de um identificador único a cada objeto e torna o acesso a estes relativamente simples. Apesar desta arquitetura estar implementada em grande escala a muitos serviços de *e-Bussiness* é de realçar que existam ainda muitos poucos casos da utilização desta arquitetura em sistemas de gestão, principalmente em sistemas de gestão de infraestrutura *Cloud*.

Neste sentido Hyuck Han et al. cria um sistema capaz de fazer esta gestão de infraestruturas *Cloud* com base na arquitetura *REST*, chamando-lhe *CMS* (*RESTful Cloud Management System*). Neste artigo é demonstrado como os elementos geridos se podem tornar recursos *REST* e como operações existentes em sistemas podem ser avaliadas usando os quatro métodos *REST*, ou combinações entre eles. Para além disto é ainda descrito como componentes de sistemas de gestão

existentes podem ser lidos como serviços Web, em formato *REST*.

Um estudo elaborado por Giuseppina Cretella et al. [CDM12] demonstra todas as técnicas de anotação existentes para descrição semântica de *APIs* expostas como serviços *Web* através de protocolos baseados em *REST*. No cenário da *Cloud*, os requisitos funcionais, como os pedidos ao nível dos serviços, são particularmente importantes para a caraterização de serviços *Cloud*. A técnica apresentada neste artigo inclui a descrição semântica de tais requisitos para a classificação de serviços *Cloud*. Em [VVE10], Toby Velte et al., analisam toda a infraestrutura *Cloud*, bem como todos os seus componentes incluindo a arquitetura *REST* e o formato de resposta *JSON*.

A *API* desenvolvida no âmbito desta dissertação possui várias vantagens que se destacam das demais existentes, começando com o facto de ser livre (*open source*) e estar disponível para revisões e alterações. Para a partilha da biblioteca foi criado um repositório no *Github*[3] que contém toda a biblioteca desenvolvida. Nos casos de aplicações como o *Otixo*[4] ou o Jolidrive[5] o principal problema encontra-se neste fator, o não conhecimento público da *API* desenvolvida para integrar de forma uniforme todos os fornecedores implementados por eles. Ambas as aplicações implementam o protocolo de autorização *OAuth*, tal como no caso da *API* desenvolvida.

Existem ainda outras bibliotecas públicas e que lidam com sistemas de *Cloud*, como a *Delta-Cloud*[6], que implementa diversos fornecedores de serviços na Nuvem. Contudo, esta biblioteca está desenvolvida e focada para a computação na Nuvem, e não tanto para aceder aos serviços de armazenamento. São poucos os fornecedores de armazenamento suportados por esta biblioteca e todos eles encontram-se fora do pretendido para esta dissertação, pois não existe suporte para *OAuth*, quer da parte desta biblioteca, quer dos fornecedores que ela implementa, à exceção do *Google Storage*. Tendo todos estes parâmetros em consideração optou-se por desenvolver uma *API* proprietária que correspondesse diretamente às exigências e necessidades da arquitetura proposta mais à frente, intitulada de *Simploud*.

[3] https://github.com/joaojosegouveia/SimploudAPI.git
[4] http://otixo.com/
[5] https://drive.jolicloud.com/welcome
[6] http://deltacloud.apache.org/

Capítulo 3

Implementação

Neste capítulo são apresentados os dois sistemas implementados no âmbito dos objetivos propostos para esta dissertação. Na primeira parte do capítulo será estudada e aprofundada a arquitetura e sistema implementados (com o nome de *Simploud*) para permitir a autenticação e autorização de utilizadores em fornecedores de armazenamento na Nuvem externos com o Cartão de Cidadão Português. Será apresentada também uma *API* que gere de forma completamente uniforme os pedidos e respostas obtidas por parte dos fornecedores externos abordados.

O objetivo da segunda implementação é criar um serviço que permita o suporte da autenticação via *Cartão de Cidadão* em qualquer serviço que permita delegação de autenticação. Para isso foi criado um sistema que engloba duas aplicações, uma com um fornecedor *OpenID* e outra com uma Entidade que delega a autenticação ao primeiro.

3.1 Autenticação com CC via sessão *SSL*

Em ambas as implementações, o Cartão de Cidadão funciona como método de autenticação fazendo uso do seu certificado para esse efeito. Ambas as soluções criadas são desenvolvidas em linguagens *Web*, mais precisamente *ASP .Net MVC*. A principal razão de escolha de uma aplicação *Web*, em vez de uma aplicação *desktop*, centra-se no suporte dos navegadores *Web* existentes para sessões *SSL* através da troca de certificados digitais. Outra razão óbvia refere-se ao facto de ser necessário que a troca dos parâmetros do certificado de autenticação do CC se processe dentro de uma sessão *SSL*, bem como todos pedidos e respostas obtidas do protocolo *OAuth*. Por estas razões optou-se por uma aplicação assente no navegador *Web* do cliente.

3.1.1 Certificados de servidor

Primeiro que tudo é importante entender qual a diferença existente entre este tipo de certificados e os certificados de cliente. Um certificado de servidor é um certificado digital emitido para qualquer aplicação ou serviço *Web* por uma entidade certificadora confiante, conhecidas e abordadas nesta dissertação por *CA*. Um certificado de servidor verifica a identidade de uma dada organização ao cliente para que o cliente possa navegar, com segurança, através das aplicações e serviços *Web* dessa organização. Estabelece-se então uma relação de confiança em que o cliente sabe à partida que:

- as aplicações e serviços aos quais acede pertencem, de facto, a essa organização e não a um impostor;
- as transações entre servidor e cliente são cifradas.

Neste caso os certificados funcionam como a identidade das aplicações e serviços *Web* perante os navegadores dos diversos utilizadores. Eles certificam o navegador de que aquela aplicação é quem diz ser. O esquema é tão simples quanto isto, embora na realidade seja necessário

algo mais para autenticar uma entidade na *Web*. Se uma aplicação *Web* criasse o seu próprio certificado de autenticação facilmente outro indivíduo mal intencionado poderia recriar este certificado e enviá-lo para o navegador dos utilizadores, sem que estes dessem conta de que estavam a aceder a aplicações falsas. Estaríamos perante um ataque de *Phishing* clássico, onde um indivíduo mal intencionado se faz passar por quem realmente não é para obter dados pessoais de utilizadores. A resolução deste problema está na criação de Autoridades de Certificação, reconhecidas a nível mundial como entidades confiáveis e onde, como já foi dito na secção 2.3.2.2, podem delegar este nível de confiança e fiabilidade aos certificados que ela própria cria. A validação e verificação da cadeia de certificados de uma dada aplicação passa a ser feita por estas entidades, delegadas pelo navegador do cliente.

Falta então perceber como é feita a verificação destes certificados e como é que o canal estabelecido entre o servidor e o cliente é cifrado.

Cada certificado possui uma chave de conhecimento público e uma chave privada. Esta chave privada apenas é do conhecimento da entidade tutora do certificado e da *CA* que o emitiu. As transações executadas entre servidor e cliente são cifradas com base neste par de chaves. Todos os dados que forem cifrados com a chave privada do certificado do servidor, apenas poderão ser decifrados com a chave pública do mesmo, e vice-versa.

O processo de autenticação baseada em certificados é facilmente compreendido, podendo ser dividido em quatro fases:

- **Primeira Fase**: Caso a data atual não esteja em conformidade com o intervalo de tempo admitido pelo certificado, então quer dizer que o certificado se encontra fora do período de validade, o que faz com que o processo de autenticação termine imediatamente. Caso contrário o utilizador prossegue para o passo seguinte;
- **Segunda Fase**: Cada cliente *SSL* habilitado mantém uma lista de certificados de autoridade de certificação confiáveis. Esta lista determina quais os certificados de servidor que o cliente aceitará. Se o nome distinto (DN) da autoridade de certificação emissora coincide com o nome distinto da autoridade de certificação na lista de autoridades de certificação confiáveis do cliente, então a *CA* é uma autoridade de certificação confiável e o cliente procede para a próxima fase. Se a *CA* não estiver na lista, o servidor não é autenticado, a menos que o cliente encontre e possa verificar uma cadeia de certificados, terminando com uma autoridade de certificação que está na lista;
- **Terceira Fase**: O cliente faz uso da chave pública do certificado da autoridade de certificação (que deverá encontrar na lista de *CAs* confiáveis) para validar a assinatura digital da autoridade de certificação no certificado do servidor. Se as informações no certificado do servidor foram alteradas desde que este foi assinado pela autoridade de certificação, ou caso a chave pública do certificado de autoridade de certificação não corresponda à chave particular que foi usada pela *CA* para assinar o certificado de servidor, o cliente não autentica a identidade do servidor. Se a assinatura digital da autoridade de certificação pode ser validada, o cliente prossegue. Neste ponto, o cliente determinou como válido o certificado do servidor.
- **Quarta Fase**: Apesar deste passo não ser obrigatório, nem constar nas especificações do protocolo *SSL*, serve para evitar um conhecido problema criptográfico, dado pelo nome

de *Man-In-the-Middle*, ou Homem-no-Meio. Para evitar este ataque, o cliente apenas necessita verificar se o nome de domínio no certificado do servidor corresponde ao nome de domínio do próprio servidor. Caso estes nomes não coincidam, o cliente deverá recusar a ligação, pois poderá estar sob este tipo de ataque.

No final deste processo o cliente e servidor estão aptos a comunicarem entre si de forma segura, tendo o cliente a certeza da identidade do servidor, através do seu certificado.

3.1.1.1 Criação da cadeia de certificados de servidor

Para a criação da cadeia de certificados de servidor foi utilizado o software *XCA* (*X Certificate and key management*), que permite a criação e gestão de certificados X.509, pedidos de certificados, chaves privadas *RSA*, *DSA* e curvas elípticas, *Smartcards* e *CRLs*. Tudo o que for necessário para uma Autoridade Certificadora é implementado neste programa. Todas as *CAs* podem assinar *CAs* de camadas inferiores recursivamente, possibilitando a criação de cadeias de validação de certificados. O uso deste *software* foi importante exatamente para esse processo, a criação do certificado de servidor utilizado, bem como da cadeia de certificados que valida o próprio.

A cadeia construída para a validação de certificados segue a estrutura da Figura 3.1, onde foram criados três certificados por esta entidade: o certificado da CA, o certificado assinado por esta e utilizado como certificado de servidor para a aplicação *Simploud* e ainda o certificado de servidor, igualmente assinado por esta CA, para o servidor *OpenID*.

A implementação desta estrutura de certificação permite que a entidade emissora criada consiga gerir todos os certificados assinados por ela. Desta forma, para além de podermos ter um número de certificados ilimitados e totalmente controlados por esta entidade, podemos ter vários servidores a correr as aplicações de forma distribuída. Sempre que um certificado for comprometido ou simplesmente revogado, a aplicação continuará a correr através de outros servidores, assentes em certificados idênticos.

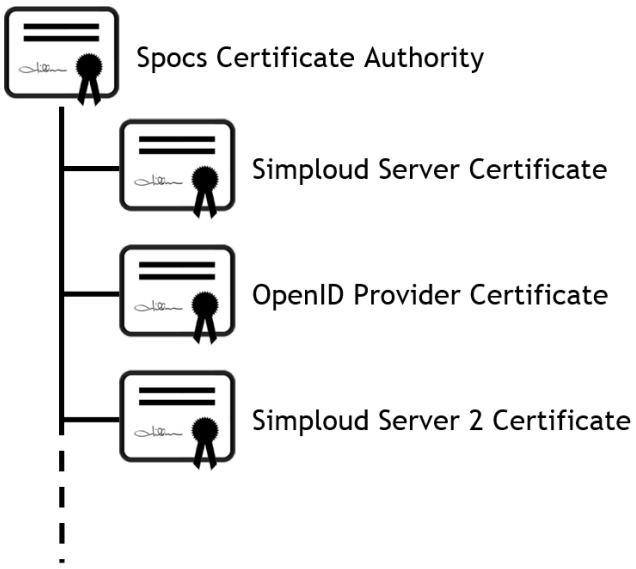

Figura 3.1: Cadeia dos certificados de chave pública construída para as arquiteturas *Simploud* e *OpenID Provider*

Para a criação dos certificados é necessário criar uma base de dados local, acedida pela própria aplicação, para guardar todo o trabalho desenvolvido. Como esta base de dados guarda informações importantes, e é necessário que ninguém para além do criador da mesma aceda a ela, é guardada localmente completamente cifrada.

Posto isto, para a criação do certificado de raiz da CA basta configurar o *software* XCA da seguinte forma:

- Clicar no botão *New Certificate*;
- Definir o formato do certificado a ser criado como *[default] CA*;
- Selecionar o separador *Subject*:
  - Escolher um *Internal Name*, nome com a função de identificar o certificado na ferramenta;
  - Preencher os dados referentes ao *Distinguished Name*, onde o *Common Name*, no nosso caso, foi *Spocs Certificate Authority*. Caso se pretenda podem ser adicionados parâmetros adicionais a este campo;
  - Selecionar a chave privada desejada, podendo-se optar pelos tamanhos de 1024, 2048, 4096 bits e pelos algoritmos criptográficos RSA, DSA ou Curva Elíptica.
- Clicar de seguida no separador de extensões e definir o parâmetro de *Time Range*. Um valor aceitável para este intervalo de validade deverá ser de dez anos;
- Por fim é importante definir um *URL* para a CRL da CA, algo do género *http://spocs.it.ubi.pt/crl/crl.der*.

Para a criação do certificado de servidor é necessário que se sigam os mesmos passos, com pequenas alterações:

- O formato a utilizar é *[default] HTTPS_server*;
- Na secção *Signing*, é necessário escolher o certificado que vai assinar o novo certificado a ser criado;
- O parâmetro *Common Name* deverá conter o caminho especificado para o site, pois considera-se uma boa prática. Assim sendo deverá ser algo como *spocs.it.ubi.pt:444/simploud*;
- Caso exista uma CRL (criada no primeiro certificado), o campo da CRL deverá ser preenchido com o mesmo endereço colocado em cima.

Posto estes dois processos executados, deverá estar criada a estrutura considerada para esta implementação, com um certificado de raiz da CA com o nome de *Spocs Certificate Authority* e um certificado de servidor intitulado de *spocs.it.ubi.pt:444/simploud*, assinado com a chave privada da CA referida acima.

3.1.2 Certificados de cliente

Em muitos casos de implementações, é necessária a autenticação através de certificados de cliente, para que o cliente prove a sua identidade ao servidor e assim se estabeleça uma ligação de confiança mútua entre ambas as partes.

Na secção anterior falou-se de certificados de servidor, onde o cliente procura autenticar a identidade do servidor. Mas desde sempre as aplicações e serviços *Web* precisaram de métodos de autenticação dos seus utilizadores. Um dos mecanismos que mais benefícios traz aos sistemas desenvolvidos pela internet fora baseia-se em certificados de cliente. Este processo consiste, ao contrário dos certificados de servidor, na necessidade do servidor autenticar a identidade do cliente. Este processo traz enormes vantagens e exemplos disso são:

- através de sessão *SSL*, fazer autenticação em aplicações e serviços *Web*;
- permitir autenticação multi-fator (uma aplicação *Web* poderia pedir informação que apenas o utilizador conhece, por exemplo palavra-passe, e o que possui, como o certificado).

Atualmente é possível criar nestes serviços uma instrução de obrigação para que quem acede a eles necessite apresentar um certificado digital, algo que apenas este indivíduo possui. Seguindo este raciocínio foi implementada esta função na aplicação desenvolvida, para que apenas utilizadores titulares de cartões *Smartcard* se possam autenticar e utilizar esta aplicação.

Para implementar a autenticação forte através do certificado de autenticação do Cartão de Cidadão foi necessário proceder a algumas adaptações. De modo a que o servidor efetue o pedido do certificado existente no Cartão de Cidadão, são necessários diversos passos, apresentados de seguida.

3.1.2.1 Instalação de certificados de raiz do *CC*

De modo a ser validada a totalidade da cadeia de certificação do certificado presente no Cartão de Cidadão, é necessário adicionar os respetivos certificados à loja de certificados do *Windows*, sistema operativo que se encontra instalado no servidor onde foi instalada a aplicação *Web*.

Para se proceder à instalação de todos os certificados do Cartão de Cidadão basta fazer a transferência dos certificados[1] e colocá-los dentro da pasta *Trusted Root Certification Authorities*.

Uma das formas possíveis de validar que os certificados ficaram corretamente instalados, é abrindo o certificado cliente na máquina onde está a ser efetuada a instalação. O certificado não deverá apresentar nenhum erro ou aviso, sendo que a cadeia de certificação deverá aparecer com a indicação de que o estado de cada certificado é *OK*.

3.1.2.2 Configurar o servidor para ligações *SSL*

Configurar o servidor para ligações *SSL* é acima de necessário, obrigatório, não só por causa do uso de certificados digitais mas também porque todo o protocolo *OAuth* assim o obriga. Este processo é simples e apenas requere que aquando da publicação da aplicação Web no servidor

[1] Certificados do Cartão de Cidadão disponíveis em http://www.cartaodecidadao.pt/index.php%3Foption=com_content&task=view&id=113&Itemid=100&lang=pt.html

Spocs, pertencente ao Instituto de Telecomunicações da Universidade da Beira Interior, seja criada a cadeia de certificados referida em 3.1.1.1 e que a aplicação tenha como certificado para criação de sessão *SSL* o certificado de servidor dessa cadeia. No *IIS*, este processo requere a criação de um novo *website*, e ligar o suporte para sessão *SSL*, ao selecionar a opção *Bindings*, depois adicionar uma nova ligação com o tipo definido como *HTTPS* e colocar o certificado de servidor *SSL* criado para o efeito.

3.1.2.3 Configurar o servidor para aceitar certificados digitais de clientes

A configuração do servidor para aceitar certificados digitais de clientes pode ser efetuada através da alteração do ficheiro de configuração da aplicação *Web.config* servindo para isso colocar as linhas de código apresentadas em 3.1. Outra possibilidade para alterar esta configuração é, selecionar a aplicação no gestor do *IIS*, clicar de seguida em *SSL Settings* e marcar a opção para requerer *SSL* e não esquecer que é necessário colocar o estado da variável *Client Certificates* como *Required*.

```
<security>
   <access sslFlags="Ssl, SslNegotiateCert" />
</security>
```

Código 3.1: Alteração da configuração para permitir a aceitação de certificados de cliente

3.1.2.4 Configurar o servidor para aceitar o certificado do Cartão de Cidadão

Os servidores estão normalmente configurados para pedir e aceitar certificados clientes que sejam emitidos pelo seu *LDAP* (*Lightweight Directory Access Protocol*). O manual técnico do Cartão de Cidadão indica que se deva configurar o servidor de modo a aceitar igualmente certificados emitidos pela entidade certificadora emissora dos certificados presentes no Cartão de Cidadão, mas através do código anterior qualquer tipo de certificado de cliente instalado na loja de certificados pode ser escolhido pelo utilizador.

3.1.2.5 Validação aplicacional do certificado

De forma a garantir que só são aceites certificados presentes nos Cartões de Cidadão, o código desenvolvido para autenticação deve validar um conjunto de parâmetros presentes no certificado, de forma a garantir a origem do certificado. Para efetuar esta validação é necessário efetuar todas as verificações abordadas na secção 3.1.1 e ainda garantir que apenas os certificados do Cartão de Cidadão podem autenticar-se neste serviço, tendo que se recorrer a uma função de filtragem construída a nível aplicacional (ver Anexo A.2.1), através da análise dos parâmetros que constituem o Nome Distinto do certificado (DN).

3.1.2.6 Validação de validade do certificado

A última validação é efetuada pela entidade emissora do certificado, de modo a garantir que o certificado não foi revogado. Para isso a aplicação Web envia um pedido de verificação para a *CA* do certificado e esta verifica a sua validade, bem como suspensão ou revogação através do recurso à sua *CRL*. O esquema 3.2 é representativo de como se processa a troca de informações entre cliente, aplicação e *PKI* do estado, ao longo do processo de autenticação do cliente que tenta aceder à aplicação.

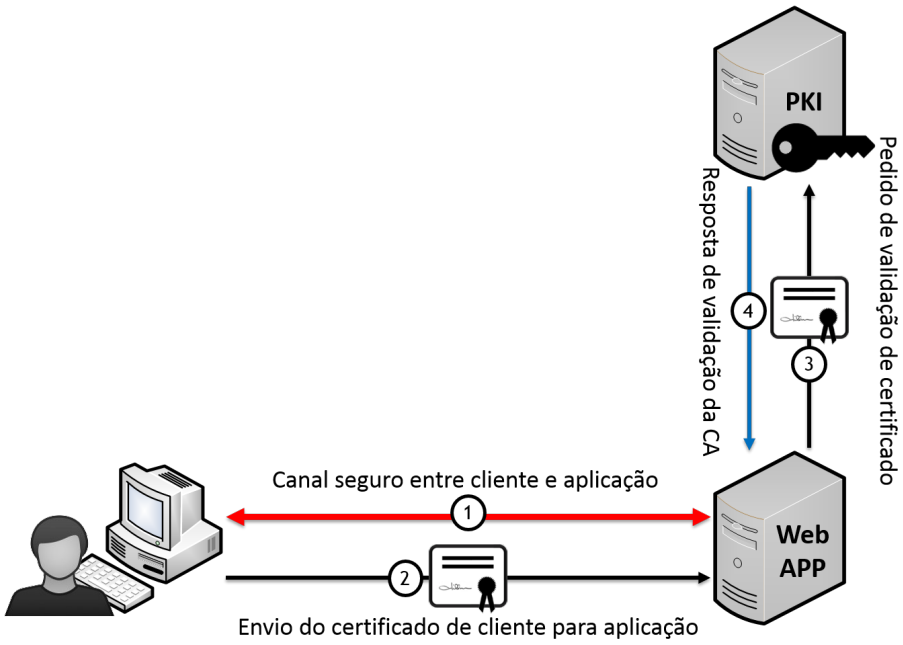

Figura 3.2: Esquema de utilização do Certificado de Autenticação

3.2 Autenticação com CC em fornecedores de armazenamento na Nuvem

A arquitetura deste sistema tem como objetivo principal aumentar os padrões de segurança atualmente utilizados pelos principais fornecedores de armazenamento na Nuvem existentes, como o caso do *Dropbox*, *MEO Cloud* e *Skydrive*. Em todos estes serviços, e seguindo a principal tendência da Web nos dias de hoje, todos estes serviços têm como método de autenticação o par nome de utilizador e palavra-passe. Com a evolução da internet é essencial que a segurança dos sistemas, aplicações e serviços que estão assentes nela, evolua lado a lado com esta tendência de exponenciação de serviços existentes.

Seguindo esta ideia e olhando para a atual conjetura da Web percebemos que o foco principal desta dissertação deve ser as infraestruturas na Nuvem, não só porque existe uma evolução clara neste sentido, como cada vez mais os utilizadores adotam estas soluções para guardar os dados. Desta forma, a implementação do sistema *Simploud* foca-se nos fornecedores de armazenamento na Nuvem mais conhecidos e utilizados para guardar recursos, como o caso da *Dropbox*, *MEO Cloud* e *Skydrive*, casos de estudo desta dissertação.

Para estes fornecedores é essencial, para além das condições e capacidades de armazenamento oferecidas aos utilizadores, que existam fortes políticas de segurança e privacidade, pois convém dizer que não é fácil para todos os utilizadores confiar os seus dados privados a estas entidades. Mais difícil ainda será convencer outras empresas, com dados privados que podem comprometer o seu próprio negócio, a guardar os seus dados na Nuvem. Neste sentido o *Simploud* agrega, numa primeira fase, todos os três fornecedores anteriormente falados e aproveita a implementação do protocolo *OAuth* por parte destes serviços para criar um novo método de autenticação, que alia ao *OAuth* a autenticação forte disponibilizada pelo Cartão de Cidadão.

3.2.1 API uniforme para interação com os diferentes fornecedores de armazenamento

Ao criar uma aplicação que tem como função principal agregar num único ponto de acesso diferentes fornecedores de armazenamento na Nuvem é normal que se encontrem em pedidos semelhantes (obter um ficheiro por exemplo), diferenças notórias, pois cada fornecedor tem as suas próprias implementações dos mecanismos de acesso e autorização. Desta forma foi desenvolvida uma *API* para fazer a agregação dos diferentes fornecedores e interagir de forma uniforme com as suas *APIs* baseadas em *REST*. Como explicado anteriormente, cada fornecedor implementa as suas próprias versões do protocolo *OAuth*, define os seus objetos e formatos *JSON* devolvidos nas respostas. De facto, a escolha dos fornecedores não incidiu apenas no facto de que são os fornecedores mais conhecidos e utilizados pelos utilizadores, mas também porque cada um deles implementa uma versão diferente do protocolo *OAuth*. Assim sendo, podemos afirmar que para além de termos uma *API* que suporta vários fornecedores de armazenamento, existe ainda um suporte extensivo ao protocolo *OAuth*.

Todos estes parâmetros que diferenciam as diferentes *APIs* significam em pequenas diferenças que tornam a tarefa de criar uma *API* cuidadosa e trabalhosa, mas necessária.

O desenvolvimento da *API* é separada em dois módulos distintos: um módulo destinado para as transações do mecanismo *OAuth* e outro que executa todos os pedidos relacionados com a gestão dos ficheiros e pastas do utilizador.

O primeiro módulo foi construído com o objetivo de implementar todos os métodos e passos para a obtenção de um *Token* de autorização, recorrendo ao protocolo *OAuth*. Para apoiar esta implementação usou-se uma adaptação da classe *OAuthBase.cs* (Anexo A.3.1), disponibilizada pelos criadores deste mecanismo e que gere a criação de uma assinatura correspondente ao pedido efetuado, um *Timestamp* e um *Nonce*. Estes parâmetros têm como função definir um intervalo de tempo aceitável para a aceitação de um determinado pedido e atribuir a esse pedido um código único, respetivamente. Esta adaptação da classe de base fornecida pelo *OAuth* foi necessária para implementar o parâmetro de verificação *oauth_verifier* respeitante ao fornecedor *MEO Cloud* e também a necessidade de gerar assinaturas de pedidos que contivessem o parâmetro de redirecionamento junto com os parâmetros desses pedidos. Para a *Dropbox*, que implementa a versão 1.0 do *OAuth*, o pedido de redirecionamento é feito durante o envio do pedido de autorização (Anexo A.3.2, método *GetAuthorizeUri*), com o redirecionamento a ser incluído num dos parâmetros do pedido. Já no caso do *MEO Cloud* o pedido de redirecionamento entra como parâmetro no primeiro passo do protocolo, no pedido de obtenção do *Request Token* e este parâmetro adicional entra no processo da geração de assinatura para esse pedido. Daí a necessidade de adaptação desta classe previamente fornecida para este caso específico.

O módulo de *OAuth* foi totalmente implementado, contendo as diversas fases de autorização deste protocolo, como de resto se pode verificar no Anexo A.3.2. Primeiro são definidas as variáveis dos endereços finais, variáveis essas que complementam todos os *URLs* de pedidos feitos à API do fornecedor. Os métodos *GetRequestToken*, *GetAuthorizeUri* e *GetAccessToken* são implementados de seguida e implementam todo o mecanismo *OAuth* para obtenção de autorização. O primeiro destes três envia um pedido para a obtenção de um *Request Token* e a resposta é guardada no objeto *Token* (Código 3.2), criado para o efeito. De seguida é enviado um

pedido para obter autorização por parte do proprietário dos recursos, sendo para isso necessário que este se autentique com as suas credenciais no *Website* do fornecedor e autorize o acesso ao consumidor de recursos, neste caso a aplicação *Simploud*. O passo final diz respeito ao envio do pedido de *Access Token*, onde o utilizador irá receber um *Token*. O método referente ao pedido de assinatura foi criado exatamente para assinar os diferentes pedidos especificados anteriormente.

```
public class OAuthToken
{
    public OAuthToken(string token, string secret)
    {
        Token = token;
        Secret = secret;
    }
    public string Token { get; private set; }
    public string Secret { get; private set; }
}
```

Código 3.2: Classe OAuthToken criada para guardar todos os Tokens obtidos

Existem algumas diferenças a enunciar durante todo este processo em relação aos diferentes fornecedores considerados:

- introdução do parâmetro *oauth_verifier* para a implementação da *API* do fornecedor *MEO Cloud*, devido à utilização da versão 1.0a do protocolo *OAuth*, enquanto que as versões implementadas para a *Dropbox* e *Skydrive* são, respetivamente, as versões 1.0 e 2.0. É ainda de referir que em Julho de 2013, a *Dropbox* implementou a versão 2.0 do *OAuth*, continuando a suportar a versão anterior;
- o modo de redirecionamento para a aplicação é diferente nos três fornecedores: na *Dropbox* e *Skydrive* o redirecionamento é efetuado junto com o pedido de autorização, logo não requer assinatura, enquanto que na *MEO Cloud* o parâmetro de redirecionamento é feito junto do pedido de *Request Token*, sendo assim necessário adaptar a classe *OAuthBase.cs* para permitir assinar pedidos que contenham mais este parâmetro;
- no *Skydrive* o método *RequestToken* não é implementado pois o mecanismo de autorização envia este objeto junto com a resposta ao pedido de autorização;
- no *Skydrive*, a obtenção do *Access Token* processa-se através do método *HTTP POST* e o conteúdo deve estar especificamente formatado sob a forma *application/x-www-form-urlencoded* e a resposta obtida, em formato *JSON* é desserializada para o objeto *AccessToken* A.3.4, enquanto que na *Dropbox* e *MEO Cloud* o método *HTTP* utilizado é *GET* e usa-se o objeto *OAuthToken* para guardar a resposta obtida.

Concluído o processo de autorização é então permitido ao utilizador, através do uso da aplicação *Web*, gerir da forma que bem entender o acesso que este detém sobre os recursos. Desta forma, foi necessária a implementação de um segundo módulo na *API* que consiste no tratamento uniforme dos pedidos feitos aos diferentes fornecedores. Existem também neste módulo diversas diferenças entre fornecedores, enumeradas de seguida:

- **Dropbox**: Implementa três tipos diferentes de respostas *JSON*, uma para caraterizar os detalhes de conta de utilizador, outra para informação acerca de ficheiros e por último

uma para diretorias. A diferença entre os objetos *File* e *Folder* é o atributo *contents* que está presente apenas no objeto *Folder* e representa a listagem de conteúdos de uma diretoria (Anexo A.3.4). A especificação de acesso a conteúdos do utilizador é feita através da notação de caminho, sendo única para cada conteúdo;

- **MEO Cloud**: Os formatos *JSON* atualmente utilizados por este fornecedor são idênticos aos usados pela *Dropbox*, bem como os objetos considerados, um tipo para ficheiros e outro para diretorias;

- **Skydrive**: Implementa a versão 2.0 do *OAuth* e faz uso de todas as alterações associadas à versão deste protocolo, como a implementação do conceito de *Token* de refrescamento e escopos. Quando o protocolo se inicia é necessário especificar quais os escopos a serem utilizados nesta aplicação (consideramos ser apenas necessários os escopos *wl.basic* e *wl.skydrive_update*; o primeiro serve para retirar informações do perfil de utilizador, sendo que no caso desta aplicação apenas serve para retirar informação acerca do espaço de armazenamento ocupado e disponível; o segundo permite acesso à leitura e escrita em ficheiros guardados no *Skydrive*). Após este procedimento, a aplicação está apenas autorizada pelo utilizador a aceder aos escopos designados. Com esta versão do *OAuth*, e olhando particularmente para este fornecedor, os *tokens* de acesso obtidos contêm uma data de expiração, de uma hora, e no final deste período é necessário refrescar o acesso do consumidor ao serviço. Utilizando o mecanismo de refrescamento do protocolo *OAuth* é possível obter um novo *token* sem que o utilizador se aperceba deste processo. Sobre as respostas *JSON*, o *Microsoft Live Connect*, *API REST* do *Skydrive* possui objetos para *Activity*, *Album*, *Application*, *Audio*, *Calendar*, *Comment*, *Contact*, *Error*, *Event*,

  File, *Directory*, *Friend*, *Permission*, *Photo*, *Quota*, *Tag*, *User* and *Video*. Apesar da existência de todos estes objetos, apenas alguns deles são necessários para o *Skydrive*. A *API* desenvolvida apenas implementa os objetos *File*, *Directory*, *Quota* e *User*, pois estes são os necessários para o funcionamento mínimo da aplicação. Os últimos dois são necessários para extrair o *email* do utilizador autenticado e também informação acerca do espaço de armazenamento ocupado e disponível, guardados na estrutura *Quota* e *Account* (ver Anexo A.3.4 para mais detalhes). O último detalhe que distingue este fornecedor dos anteriores é o acesso aos conteúdos, pois em vez de fazer uso de especificação de caminhos, cada objeto de ficheiro ou pasta possui um parâmetro *id* que representa o seu identificador único.

Apesar de todas as diferenças especificadas todos eles possuem semelhanças em alguns aspetos. Por exemplo, apesar da estrutura dos pedidos *URL* ser diferente para cada fornecedor, os passos que compõem o processo de autorização até ao acesso final aos recursos protegidos são idênticos, tal como todos os fornecedores implementam o mesmo tipo de funções e operações sobre ficheiros: *Download*, *Upload*, listagem, partilha, cópia, renomear, histórico de versões para recuperação de ficheiros, mover e apagar. A *API* desenvolvida implementa praticamente todas estas operações, à exceção da partilha de conteúdos com outros utilizadores e a recuperação de versões anteriores, como se pode verificar pelo exemplo do Anexo A.3.3, onde estão implementadas os métodos:

- *GetAccountInfo* para obter informação acerca da conta do proprietário dos recursos;

- *GetFiles* para obter o conteúdo de uma pasta, e consequentemente listá-lo;

- *CreateFolder*, *Rename* e *Delete* para criar uma pasta, renomear e remover um ficheiro ou pasta, respetivamente;
- *DownloadFile* e *UploadFile* para transferência e carregamento de ficheiros.

3.2.2 Arquitetura do sistema *Simploud*

A arquitetura do sistema tem em consideração os objetivos principais deste projeto e, desta forma, podemos considerar quatro principais entidades a considerar para a realização dos objetivos propostos:

- **Cartão de Cidadão**: é talvez a entidade central dentro da arquitetura proposta. No *Simploud* o CC tem como principal função o uso do certificado de autenticação para que o titular do cartão se possa autenticar no sistema;
- **Aplicação**: A aplicação *Web* construída tem como principal função fazer a ligação entre o Cartão de Cidadão e o ambiente *Cloud*. É função desta entidade garantir a validação da autenticação e, posto isto, gerar os *tokens* de acesso aos recursos protegidos e também executar todos os pedidos às *APIs* dos fornecedores de armazenamento na Nuvem;
- **API**: Foi necessário para esta implementação criar uma *API* que possibilitasse o acesso uniforme a todos os fornecedores implementados. É normal que, apesar de existirem normas e definições protocolares obrigatórias a seguir, cada fornecedor implemente próprias definições de objetos, atributos e haja diferenças na utilização dos protocolos implementados. Para normalizar estas disparidades aos olhos de um programador adicionou-se esta camada adicional que estabelece a ligação entre a aplicação proprietária e os fornecedores de armazenamento na Nuvem;
- **Ambiente *Cloud***: Esta entidade representa todos os fornecedores de armazenamento na Nuvem, todos os seus protocolos, métodos, objetos e atributos. É a *API* desenvolvida que estabelece a ligação com esta camada e gere os pedidos feitos e respostas obtidas à Nuvem.

O esquema geral da arquitetura do sistema apresentado na Figura 3.3 relaciona as quatro entidades anteriormente referidas, através de processos, pedidos e respostas entre elas.

Tudo começa com a autenticação do titular do cartão. O processo de autenticação de um utilizador está relacionado com a validação do certificado que este apresenta (1). Se o certificado não for validado, o utilizador não se consegue autenticar e o processo é interrompido. Caso o primeiro e segundo passo (2) sejam efetuados com sucesso, a aplicação vai usar dados referentes ao certificado e guardá-los numa base de dados (3). Caso, após os dados serem guardados na base de dados, o utilizador se tente autenticar com o mesmo certificado e se este for dado como revogado, expirado, suspenso ou inválido, então os dados referentes a este certificado são eliminados da mesma, afim de garantir a segurança, integridade e consistência dos dados. O principal objetivo desta base de dados é guardar dados referentes ao processo de autenticação e à geração de *Tokens* pelo protocolo *OAuth*, relacionando estas duas entidades, o cartão de Cidadão como identificador digital do utilizador e os fornecedores de armazenamento como gestores dos recursos desse mesmo utilizador. A Figura 3.4 representa o Modelo da base de dados proposta na integra.

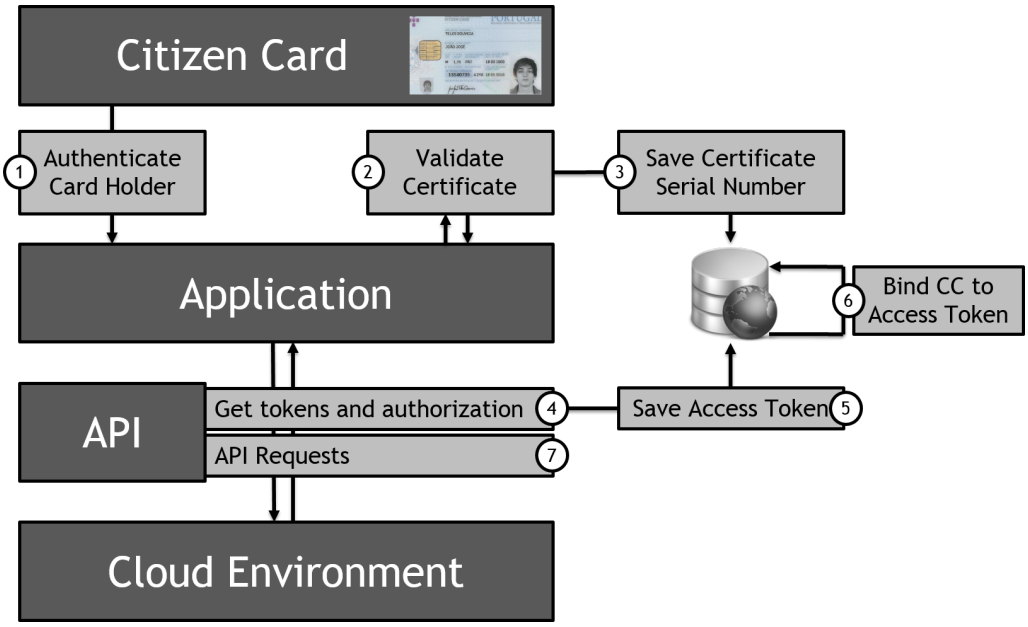

Figura 3.3: Esquema geral da arquitetura do sistema *Simploud*

Esta base de dados está dividida em três tabelas distintas:

- A tabela **Certificate** representa e guarda os dados necessários correspondentes ao certificado de autenticação do Cartão de Cidadão. Contém os atributos:
  - **IssuerName**: contém o nome do certificado e o nome do emissor;
  - **SerialNumber**: representa o número de série do certificado. Juntamente com o *IssuerName* formam a chave primária da tabela, servindo de identificador único do certificado;
  - **PublicKey**: é ainda guardada a chave pública do certificado, embora não seja aplicada em nenhum contexto específico.
- A tabela **Cloud** guarda os dados referentes aos diferentes fornecedores de armazenamento considerados para este sistema. Fazem parte dela os seguintes atributos:
  - **CloudID**: identificador único e também chave primária da tabela que guarda um valor para entrada na tabela;
  - **Name**: representa o nome dos fornecedores de armazenamento na Nuvem considerados.
- A tabela **Register** estabelece a ligação entre as duas tabelas anteriores, sendo que a chave primária desta tabela é uma chave conjunta das chaves primárias das tabelas anteriores. Podemos identificar como chave primária os atributos *IssuerName*, *SerialNumber* e *CloudID*. Para além destes atributos, esta tabela reserva ainda espaço para armazenar a informação relativa aos *Tokens* obtidos:
  - **AccessToken**: representa o identificador do *Token* obtido;
  - **AccessTokenSecret**: guarda a informação referente ao segredo do *Token* obtido.

Para efeitos de segurança são apenas guardados, na base de dados, os valores do *hash* dos dados referentes ao Cartão de Cidadão. O mesmo procedimento não pode ser efetuado para os dados

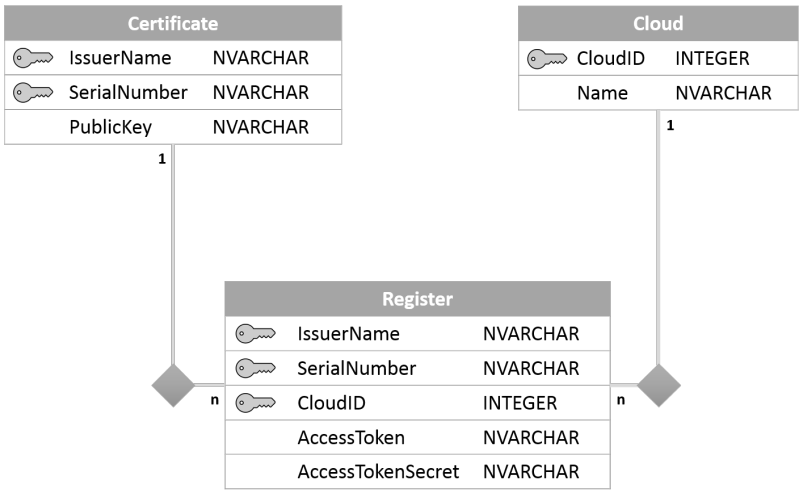

Figura 3.4: Modelo da base de dados que suporta a arquitetura *Simploud*

referentes aos *tokens* de acesso dos fornecedores de armazenamento, visto serem definitivos, exceto no caso do *Skydrive*, e ser necessário saber o seu valor real para efeitos de autorização de pedidos em próximas sessões que não a primeira. A cifra destes dados poderia ser efetuada através de mecanismos de cifra simétricos ou assimétricos, gerados pelo próprio servidor de base de dados, onde esta se encontra guardada, mas este processo não é necessário devido ao facto do potencial acesso à base de dados e consequente roubo do *token* de acesso levar a nenhuma consequência, a não ser que o atacante consiga também saber qual a chave que o consumidor (a nossa aplicação) guarda. A consequência de não guardar estes dados implicava que, a cada sessão iniciada com a aplicação, fosse necessário dar nova autorização para se aceder aos recursos protegidos na Nuvem. De qualquer das formas, é sempre assegurada a comunicação *SSL* entre cliente e servidor.

Focando de novo a arquitetura da aplicação. Ao guardar os dados referentes à autenticação da identidade do utilizador, a *API* desenvolvida tem como objetivo interagir com o protocolo *OAuth* afim de obter todos os *tokens* necessários até à obtenção do *token* de acesso, o símbolo final deste protocolo que garante acesso aos dados protegidos, com autorização do proprietário dos recursos (4). Ao obter este *token* final, que lhe permite aceder aos recursos protegidos, é importante guardá-lo na base de dados (5), pois caso contrário sempre que o utilizador criar uma nova sessão é necessário iniciar de novo o mecanismo *OAuth*. Com o *token* guardado na base de dados e também os dados referentes à autenticação procede-se de seguida à ligação entre estas duas entidades, envolvendo ainda uma variável que indica qual o fornecedor escolhido, e assim ligar o utilizador ao *token* respetivo a um dado fornecedor (6). Com o *token* guardado na base de dados e autenticação efetuada o utilizador está em condições de gerir e manipular (7) os dados protegidos na Nuvem referentes a um fornecedor de armazenamento específico.

3.2.2.1 Modelo de segurança

Os requisitos básicos de segurança que um sistema criptográfico deve fornecer são Confidencialidade, Integridade, Autenticação e Não-Repúdio. A segurança deste sistema como um todo, irá depender de cada um dos elementos que o constitui. Assim, o sistema depende do Cartão de Cidadão como objeto de autenticação de identidade eletrónica, através do certificado de autenticação que este possui, nos canais de comunicação utilizados entre o cliente e a aplicação

Web, e entre a aplicação e a base de dados, sendo que esta guarda dados sensíveis, na resistência a ataques que possam ocorrer durante o processo de autorização através do protocolo *OAuth* e ainda a segurança do esquema *IBE* (*Identity Based Encryption*) implementado para o carregamento e transferência de ficheiros cifrados para o fornecedor de armazenamento na Nuvem.

Neste tipo de sistema, em que a autenticação se processa através da troca de certificados de cliente, é importante que seja impossível para um atacante fazer-se passar por um utilizador válido, neste caso um titular do Cartão de Cidadão. Pressupõem-se que os canais de comunicação entre a aplicação e o cliente sejam seguros, e para isso todas as comunicações decorrem sobre o protocolo *SSL*, onde é impossível para um atacante obter a chave privada do certificado do servidor, estando assim impossibilitado de decifrar os dados que vão passando pelo canal. De igual forma, pressupõem-se que os canais de comunicação entre a aplicação e a base de dados sejam seguros, e que apenas os utilizadores fidedignos da aplicação (que já provaram a sua identidade) possam aceder a esta, tanto para efeito de leitura, como de escrita. Em relação ao CC, é pressuposto que um atacante não consegue alterar informação sensível guardada no *chip* do mesmo, como por exemplo o par de chaves que suportam o certificado de autenticação. É igualmente pressuposto que, durante o processo de autenticação, seja estabelecida uma relação de confiança entre a *CA* emissora do certificado do CC e a aplicação. O sistema pressupõem ainda que não existam qualquer tipo de falhas no mecanismo do protocolo *OAuth* para obtenção de autorização de acesso a recursos protegidos.

3.2.2.2 Esquematização da *Attack Tree* para esta arquitetura

As *Attack Trees* estão relacionadas com o formalismo de *Fault Trees*, usado inicialmente pela NASA (*National Aeronautics and Space Administration*) em [VDF+02], e representa um conjunto de falhas relacionadas através de expressões booleanas e que, acontecendo em sucessão poderiam provocar falhas graves determinadas na raiz dessa árvore. Esta metodologia foi trespassada para várias ciências e, atualmente, é muito utilizada em segurança para descrever esquemas que potencialmente levariam ao colapso de toda a segurança do sistema.

As *Attack Trees* fornecem a possibilidade de modelar potenciais ameaças contra os sistemas informáticos de forma metódica [Sch99]. O primeiro elemento a ser definido neste modelo é qual o objetivo do ataque. Este elemento será a raiz da árvore. Posto isto, é necessário encontrar e identificar as diferentes maneiras de conseguir este objetivo. Estas representações seguem a definição de que os *Children nodes* são condições que devem ser satisfeitas, para que o *Parent node* seja verdade. Seguindo esta nomenclatura, a árvore que demonstra os ataques possíveis para satisfazer a premissa maior deste sistema, que consiste no acesso a dados protegidos por parte de um atacante, é apresentada na Figura 3.5.
Esta árvore indica dois caminhos possíveis para satisfazer a condição principal:

- O atacante aceder à base de dados do fornecedor de armazenamento e conseguir retirar os dados de *login* de um utilizador, aliado à posse do cartão de identificação eletrónica desse mesmo utilizador, e consequentemente conhecer o PIN de autenticação deste cartão. Se estas três condições forem satisfeitas, o utilizador é capaz de gerar um *token* de acesso sem o consentimento do real proprietário dos recursos e o atacante passa a ter acesso aos dados privados deste;

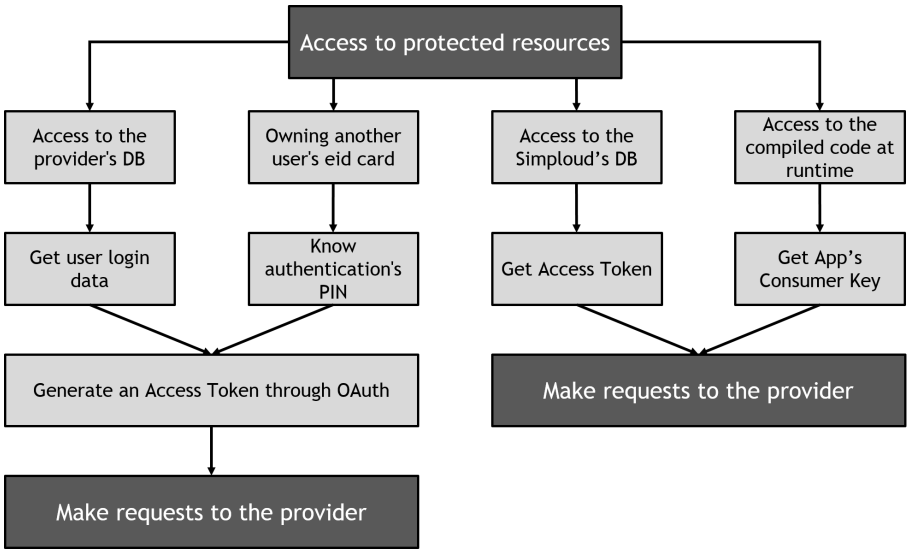

Figura 3.5: *Attack Tree* para a arquitetura proposta

- O atacante aceder à base de dados do *Simploud* e obter os *tokens* de acesso, aliado ao conhecimento da chave de consumidor, que só consegue obter através da análise do código compilado em tempo de execução. Se estas duas condições forem satisfeitas, o atacante passa a ter acesso aos dados protegidos correspondentes aos *tokens* anteriormente obtidos.

Ao identificarmos os possíveis caminhos que dariam a um atacante a possibilidade de obter acesso a dados que não lhe pertenciam (falha ao nível da Confidencialidade e Disponibilidade dos dados) foram tomadas medidas que, de certa forma, permitem reforçar a segurança com a finalidade de evitar que este cenário seja possível.

Podemos dizer que embora existam quatro caminhos possíveis, apenas dois são da nossa responsabilidade e que, representando um perigo para a segurança do utilizador foram tidas em conta as seguintes alterações:

- os dados referentes ao *token* de acesso guardado na base de dados passaram a ser guardados e cifrados com recurso ao esquema de chave simétrica implementado pelo próprio servidor de base de dados (no nosso caso o *SQL SERVER 2012*);
- utilização de técnicas de ofuscação de código para proteger o acesso às chaves de consumidor e o seu respetivos segredos (um par para cada fornecedor implementado nesta solução).

3.2.3 Integração da API desenvolvida numa aplicação proprietária

A aplicação desenvolvida para estabelecer ligação entre o Cartão de Cidadão e os fornecedores de armazenamento foi criada, como referido anteriormente, na linguagem *ASP .Net MVC* e faz uso de todos os mecanismos dos navegadores Web mais recentes para estabelecer uma sessão *SSL* com o cliente e de seguida iniciar todas as transferências protocolares até à obtenção de acesso aos recursos protegidos.

A *API* desenvolvida permite implementar fácil e uniformemente todos os pedidos executados às arquiteturas *REST* dos fornecedores de armazenamento considerados, seja durante o processo

de autorização, seja através dos pedidos para gestão de ficheiros.

Como se pode analisar através do Anexo A.1.2, apesar dos métodos *AuthorizeDropbox*, *AuthorizeCloudPT* e *AuthorizeSkydrive* corresponderem à autorização nos diferentes fornecedores considerados, os métodos da *API* chamados são iguais, mudando apenas o nome do fornecedor que estamos a considerar. O mesmo acontece com o pedido para obter o *token* de acesso e todos os outros pedidos referentes a gestão dos recursos (Anexo A.3.3).

Deste modo podemos concluir que a *API* desenvolvida satisfaz os propósitos para que foi inicialmente criada, tornando os métodos de autorização e de gestão de recursos completamente uniformes, apesar de se estar a trabalhar com diferentes fornecedores de armazenamento na Nuvem.

3.2.4 Implementação de um esquema de IBE e políticas de acesso a ficheiros

No âmbito do projeto *PRICE* surgiram vários trabalhos paralelos a este, entre eles, a implementação de vários serviços de *IBE* baseados nas propriedades criptográficas do Cartão de Cidadão. O objetivo deste trabalho consistia na criação de vários sistemas que permitissem, através de um sistema de *IBE* aliado ao processo de autenticação forte do Cartão de Cidadão, cifrar e decifrar ficheiros.

Para implementar este sistema de *IBE* nesta aplicação *Web* foi utilizada uma biblioteca desenvolvida em C#, que utiliza o esquema *Emribe* e permite implementá-lo num sistema para a cifra e decifra de ficheiros. Esta biblioteca tem como funcionalidades principais a geração de parâmetros públicos e privados, a geração de chaves privadas correspondentes a uma identidade e a cifra para múltiplas identidades e respetiva decifra de ficheiros.

A função do *PKG* é gerar os parâmetros de sistema necessários (parâmetro privado *s* e parâmetros públicos P, Q, P_{pub} e $\hat{e}(Q,P_{pub})$) e a geração de uma chave pública para cada identidade. A geração destes parâmetros pode ser efetuada com recurso às funções presentes na biblioteca:

- *byte[] GenerateMasterKey(string curve)*;
- *byte[] GeneratePparam(string curve)*;
- *byte[] GenerateQparam(string curve)*;
- *byte[] CalculatePpubparam(string curve, byte[] Pbyte, byte[] masterKey)*;
- *byte[] CalculatePairQPpub(string curve, byte[] Qbyte, byte[] Ppubbyte)*;

Após o uso destas funções a chave privada pode ser gerada, mas deve ser apenas do conhecimento do *PKG*. Todos os restantes parâmetros, são de conhecimento público e são utilizados para a cifra e decifra de ficheiros. As funções disponibilizadas para cifra e decifra tem a seguinte assinatura:

- *MemoryStream EncryptStream(string[] identities, string policy, MemoryStream streamIn, string originalFileName, string curve, byte[] paramP, byte[] paramQ, byte[] parameQPpub)*;

- *MemoryStream DecryptStream(string identity, byte[] privateKey, MemoryStream streamIn, string curve, byte[] paramPpub)*;

Como se pode verificar pela assinatura das funções, a função de cifra recebe como parâmetro uma lista de identidades, aquelas com as quais o utilizador pretende cifrar o ficheiro, sendo que no processo de decifra, e esta é uma das capacidades deste tipo de sistemas, apenas os utilizadores que se encontrem nessa lista inicial, com a qual o ficheiro foi cifrado, poderão decifrá-lo.

Aliado a isto foi ainda implementado um esquema de políticas de acesso a ficheiros que permite, para além de restringir a cifra e decifra a um grupo de identidades através do esquema IBE, limitar o acesso a estes dados cifrados através de diferentes parâmetros:

- intervalo de endereços *IP*: limitar o acesso ao ficheiro apenas para um certo intervalo de *IPs*. Se pensarmos em ambientes empresariais onde normalmente são utilizadas redes privadas, caso estejamos fora dessa rede seria impossível decifrar um ficheiro anteriormente cifrado com essa condição;
- intervalo de tempo: limitar o acesso ao ficheiro a um dado intervalo de tempo. Se pensarmos no exemplo do intervalo de tempo, estabelecido durante a cifra, entre 1/1/2020 e 1/1/2025, teríamos a certeza que o ficheiro nunca seria decifrado antes da data 1/1/2020 ou após 1/1/2025;
- nível de segurança: limitar o acesso ao ficheiro a um determinado nível de segurança. Se considerarmos que um ficheiro é cifrado com nível de segurança 1, nenhum outro utilizador fora deste valor poderá decifrar o ficheiro.

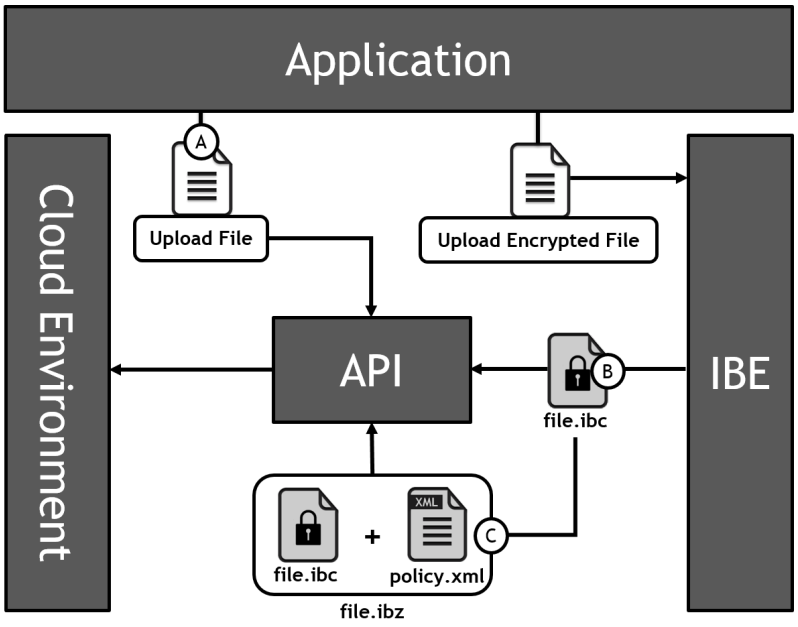

Figura 3.6: Esquema geral do processo de *Upload* de um ficheiro

A Figura 3.6 demonstra o processo de *Upload* de um ficheiro para o fornecedor de armazenamento, onde o utilizador que está a utilizar o *Simploud* passa a ter a possibilidade de enviar um ficheiro de três maneiras diferentes. Se o utilizador necessitar apenas de guardar os dados sem

a necessidade de serem cifrados, então deverá proceder à utilização da opção (A), indicada no esquema. Caso contrário, o utilizador poderá recorrer ao esquema *IBE* implementado e guardar o ficheiro de duas formas: recorrendo apenas à cifra baseada em identidade (B); utilizar a cifra baseada em identidade e acrescentar algumas políticas de acesso ao ficheiro (C). O processo da transferência de ficheiros, neste caso inverso ao envio, tem implementado igualmente estas três situações, tendo sempre a salvaguarda de que se o utilizador fizer a transferência de um ficheiro cifrado em texto limpo, sem o decifrar, o conteúdo deste nunca será percetível ao utilizador.

A utilização de políticas de acesso a ficheiros aliada ao esquema *IBE* implementado nesta aplicação Web tem como principal objetivo aumentar os padrões de segurança, nomeadamente ao nível da privacidade dos dados. O esquema montado foi pensado para ambientes empresariais, onde esta solução poderia ser uma solução realmente capaz, assente em fortes medidas de segurança e privacidade, onde esta última é a principal causa do afastamento das empresas deste tipo de sistemas de armazenamento na Nuvem.

Por último referir que o *PKG* e todo o sistema de políticas está integrado no código da aplicação. Desta forma, tanto a cifra e decifra de ficheiros como o cumprimento das políticas é da responsabilidade da própria aplicação.

3.3 Servidor *OpenID* proprietário

A segunda implementação desenvolvida no âmbito desta dissertação teve como principal objetivo a criação de um serviço que permite que utilizadores titulares do Cartão de Cidadão possam usar este *Token* como mecanismo de autenticação em qualquer aplicação ou serviço Web, desde que este suporte o protocolo *OpenID*.

O mecanismo deste protocolo já foi explicado anteriormente (secção 2.2.3), e para a sua implementação foi utilizada a biblioteca *DotNetOpenAuth*[2], disponibilizada publicamente, e adaptada conforme as necessidades para permitir a autenticação através do Cartão de Cidadão, em vez do utilizador se autenticar no fornecedor *OpenID* com o par nome de utilizador/palavra-passe.

Para melhor compreender o protocolo *OpenID* e as aplicações utilizadas da biblioteca referida em cima foi utilizado, como guia de apoio, o livro *Openid: The Definitive Guide* [RRM12].

3.3.1 Arquitetura do sistema

A biblioteca *DotNetOpenAuth* possui diversas funcionalidades, e uma delas é o suporte do protocolo *OpenID* para a linguagem *ASP .Net*. A arquitetura deste sistema utiliza dois exemplos implementados para esta linguagem:

- ***OpenIdProviderMvc***: Este projeto implementa um provedor de *OpenID*, que possibilita o registo de utilizadores, associando uma conta *OpenID* a cada registo. O principal objetivo desta aplicação Web é autenticar um utilizador em terceiras entidades de confiança, utilizando para isso o perfil utilizado.

[2] http://dotnetopenauth.net/

- ***OpenIdRelyingPartyMvc***: Este projeto simula uma aplicação Web que implementa a parte da delegação de autenticação do protocolo *OpenID*. O processo de autenticação nesta aplicação é delegado ao provedor *OpenID* e cabe a este geri-lo.

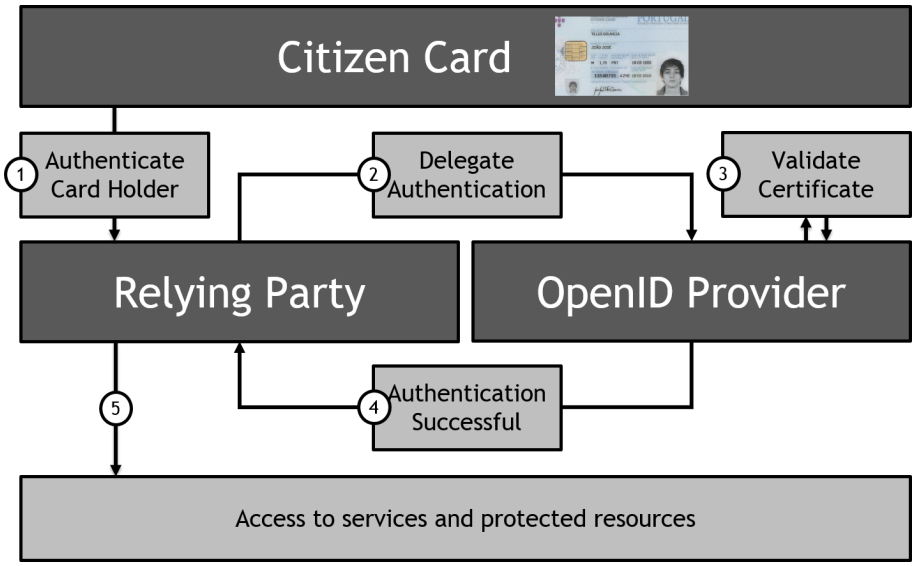

Figura 3.7: Esquema geral da arquitetura do sistema *OpenID* implementado

As principais alterações efetuadas na estrutura dos projetos já fornecidos centram-se na introdução do Cartão de Cidadão como *Token* de autenticação. Para implementar este mecanismo de autenticação foi necessário alterar ambos os projetos. No caso do provedor de autenticação foi obrigatório estabelecer uma sessão *SSL* para este comunicar com o CC. Visto que todo o código existente no processo de registo de um utilizador baseava-se no esquema nome de utilizador e palavra-passe, foi necessário alterar todo este código para substituir este mecanismo pela autenticação via Cartão de Cidadão. Para validar o certificado do Cartão de Cidadão apresentado foi implementado o código apresentado no Anexo A.1.1, bem como a função para filtragem dos certificados (Anexo A.2.1). Em relação à entidade que delega o processo de autenticação foi necessário alterar os pedidos de delegação de acesso para permitir que se mantenha a sessão *SSL*.

Para guardar os dados de autenticação para futuras utilizações foi criada uma simples base de dados, que segue o modelo da Figura 3.8. Esta base de dados é composta apenas por uma única tabela que tem como função guardar os dados relativos ao processo de registo. Os atributos que a compõem são *Username*, *IssuerName*, *SerialNumber* e *Email*. Nesta tabela apenas se considerou atribuir a chave primária ao atributo *Username* pois é este que é utilizado em todas as delegações de autenticação feitas, podendo-se classificar então como identificador único e que diferencia uns pedidos dos outros.
A Figura 3.7 representa toda a arquitetura deste sistema. Esta encontra-se dividida em três entidades principais: o Cartão de Cidadão como *Token* de autenticação, a entidade que delega o processo de autenticação e o provedor *OpenID* que gere a autenticação do utilizador. O primeiro passo consiste na tentativa do utilizador se autenticar perante uma entidade (1), onde este irá fornecer o seu identificador *OpenID*. Assim que esta entidade recebe este identificador, tenta procurar o provedor *OpenID* e assim que o encontra delega o serviço de autenticação para o provedor *OpenID* (2). Caso o certificado apresentado seja válido (3), o processo de autenticação encontra-se completado e o utilizador é redirecionado de volta para a entidade onde se quis

User		
🔑	Username	NVARCHAR
	IssuerName	NVARCHAR
	SerialNumber	NVARCHAR
	Email	NVARCHAR

Figura 3.8: Modelo de base de dados que guarda todos os dados relativos ao registo de entidades

autenticar(4). A partir daqui o utilizador tem acesso a todos os recursos e todos os serviços desta entidade (5).

3.3.1.1 Modelo de segurança

Tal como no modelo de segurança da primeira implementação, a segurança deste sistema como um todo irá depender de cada um dos elementos que o constitui. Desta forma, o sistema depende do Cartão de Cidadão como objeto de autenticação de identidade eletrónica, através do certificado de autenticação que este possui, nos canais de comunicação utilizados entre o cliente e o provedor *OpenID* e na resistência a ataques por parte do protocolo *OpenID*.

Neste tipo de sistema, em que a autenticação se processa através da troca de certificados de cliente, é importante que seja impossível para um atacante fazer-se passar por um utilizador válido, neste caso um titular do Cartão de Cidadão. Pressupõem-se que os canais de comunicação entre a aplicação e o cliente sejam seguros, e para isso todas as comunicações decorrem sobre o protocolo *SSL*, onde é impossível para um atacante obter a chave privada do servidor, estando assim impossibilitado de decifrar os dados que vão passando pelo canal. Em relação ao CC, é pressuposto que um atacante não consegue alterar informação sensível guardada no *chip* do mesmo, como por exemplo o par de chaves que suportam o certificado de autenticação. É igualmente pressuposto que, durante o processo de autenticação, seja estabelecida uma relação de confiança entre a *CA* emissora do certificado do CC e a aplicação. O sistema pressupõem ainda que não existam qualquer tipo de falhas no mecanismo do protocolo *OpenID*.

Capítulo 4

Resultados

Neste capítulo são apresentados os principais passos que envolvem o utilizador desde o ato de autenticação até ao acesso a recursos protegidos, com a exemplificação de janelas de visualização das aplicações criadas.

Na primeira secção demonstra-se todos os passos, em ambiente gráfico, que envolvem o utilizador desde o processo de autenticação até ao exemplo do envio de ficheiros para um dos fornecedores de armazenamento.

Na segunda parte deste capítulo é abordado a implementação da aplicação que delega a autenticação a um servidor *OpenID* proprietário, com exemplos de imagens retiradas da interface destas duas aplicações. É acompanhado todo o processo desde a autenticação no provedor *OpenID* até à autenticação do utilizador na aplicação que delega este processo.

Para ambos os casos serão ainda apresentados alguns testes que provam, essencialmente, que a autenticação através de cartões *Eid* não é despropositada e que o tempo de execução deste mecanismo é inferior, comparado com o tempo de execução do mecanismo de autenticação baseado em nome de utilizador/palavra-passe.

4.1 Aplicação Web *Simploud*

Como referido anteriormente, esta aplicação teve como principal fundamento a necessidade de aliar o mecanismo de autenticação forte do Cartão de Cidadão à gestão de recursos armazenados em fornecedores *Cloud*.

Assim sendo, existe a necessidade do utilizador se autenticar perante a aplicação *Simploud*, sendo para isso necessário que o utilizador selecione o certificado de cliente que pretende utilizar para autenticação (Figura 4.1), com a garantia prévia de que apenas os certificados de cliente do Cartão de Cidadão serão aceites e consequentemente validados. A seleção e correta introdução do PIN que protege o acesso ao certificado de autenticação do Cartão do Cidadão faz com que a aplicação aceite este certificado e, consequentemente, o valide. As janelas na parte inferior da Figura 4.1 demonstram os resultados visuais dos diferentes erros obtidos caso o processo de autenticação falhe. À esquerda, a janela à qual o utilizador tem acesso caso selecione o certificado errado. À direita, o erro mostrado se o certificado de autenticação do Cartão de Cidadão estiver revogado, suspenso ou expirado. Em ambos os casos o acesso à fase de autorização é interdito.

A conclusão bem sucedida do processo de autenticação resulta na janela exemplo mostrada na Figura 4.2, onde é mostrado o nome completo do titular do cartão e, em baixo, uma listagem com os diferentes fornecedores de armazenamento considerados para este trabalho. A função

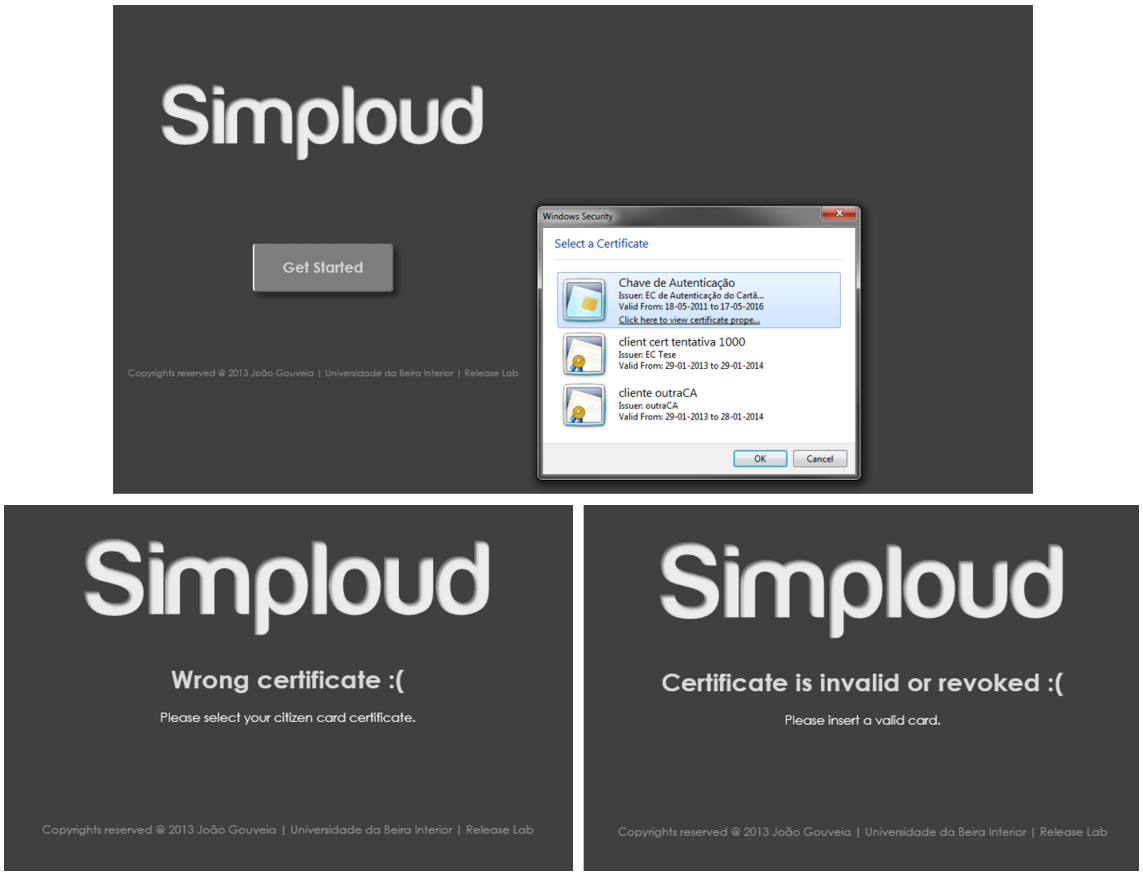

Figura 4.1: Em cima, janela de autenticação do utilizador na aplicação *Simploud*. Na parte inferior, erros de autenticação obtidos.

desta janela é permitir que o utilizador escolha o fornecedor ao qual pretende ter acesso. Ao clicar em qualquer um dos ícones, em representação do seu próprio fornecedor de armazenamento, o utilizador inicia imediatamente o processo de autorização *OAuth*. Como sabemos pelo funcionamento deste mecanismo, o pedido de autorização é efetuado na aplicação Web do próprio fornecedor, com a necessidade de autenticação do proprietário dos recursos, que em teoria deverá ser a mesma entidade que o utilizador autenticado. Deste modo, o utilizador da aplicação é redirecionado para o *Website* do fornecedor para o proprietário dos recursos dar permissão ao utilizador, e este aceder aos dados protegidos.

A conclusão deste processo com ou sem sucesso envia o utilizador de volta para aplicação *Simploud*, sendo que o resultado da autorização influencia o resultado final mostrado ao utilizador. A autorização bem sucedida resulta na Figura 4.3, com toda a informação acerca dos recursos protegidos no fornecedor anteriormente selecionado, enquanto que o contrário redireciona o utilizador para a página exemplificada na Figura 4.2. Esta janela pode ser dividida em cinco diferentes zonas:

- A **informação dos dados da conta**, onde é mostrado ao utilizador o nome da conta à qual estamos a aceder, assim como o espaço de armazenamento disponível e utilizado;
- O **Menu** principal, que contém diversos ícones para melhorar a experiência do utilizador na aplicação e permitir a fácil gestão dos seus recursos. Este menu encontra-se dividido em quatro ícones que efetuam operações diferentes:

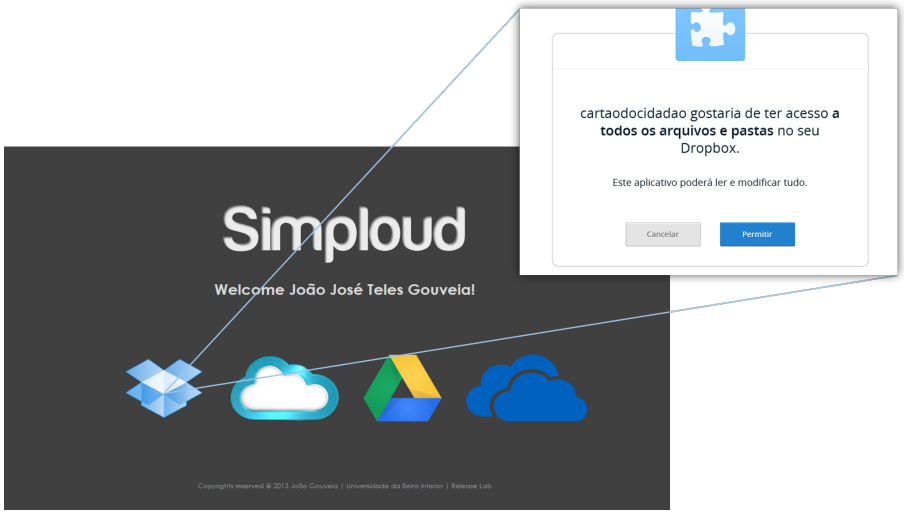

Figura 4.2: Janela para escolha do fornecedor de armazenamento pretendido

  - O primeiro ícone permite que se volte sempre à diretoria raiz do fornecedor;
  - O segundo ícone permite que se efetue o envio de ficheiros para a pasta que está atualmente em visualização;
  - De forma idêntica, o terceiro ícone trata o envio de ficheiros, neste caso particular cifrados, para a diretoria atualmente em visualização;
  - Finalmente, o último ícone possibilita que se crie uma nova diretoria na pasta atualmente em visualização.

- A **lista de recursos**, onde são mostrados todos os conteúdos (ficheiros e diretorias) que se encontram dentro da diretoria atualmente em visualização. O acesso às funções para edição de ficheiros e diretorias é feito através de uma janela de *popup*, que aparece assim que o utilizador clique em cima de qualquer ficheiro ou diretoria. O duplo clique numa diretoria envia o utilizador para a janela de visualização do conteúdo desta. Neste caso, esta diretoria que foi clicada será adicionada à lista de navegação;

- A **lista de navegação**, onde é mostrado ao utilizador todo o caminho percorrido. Para voltar a uma diretoria anterior, basta que o utilizador clique em cima do nome da pasta em questão;

- Finalmente, é apresentada novamente a **listagem de todos os fornecedores** de armazenamento implementados nesta aplicação para que, através de um simples clique em qualquer um deles, o utilizador possa alterar o fornecedor ao qual está a aceder.

Como referido anteriormente, o clique em qualquer um dos recursos visualizados permite que seja mostrada uma janela de *popup* que contém as restantes operações suportadas pela aplicação (Figura 4.4). Como deve ser óbvio, a *popup* é diferente consoante o tipo de conteúdo. Desta forma temos:

- *Popup* para diretorias: contém as funções de partilha, renomear, mover e remover, embora as funções de partilha ainda não se encontrem implementados na API desenvolvida;

- *Popup* para ficheiros: contém as funções de transferência (é possível transferir um ficheiro em texto limpo ou recorrendo a decifra através dos algoritmos IBE), partilha, renomear,

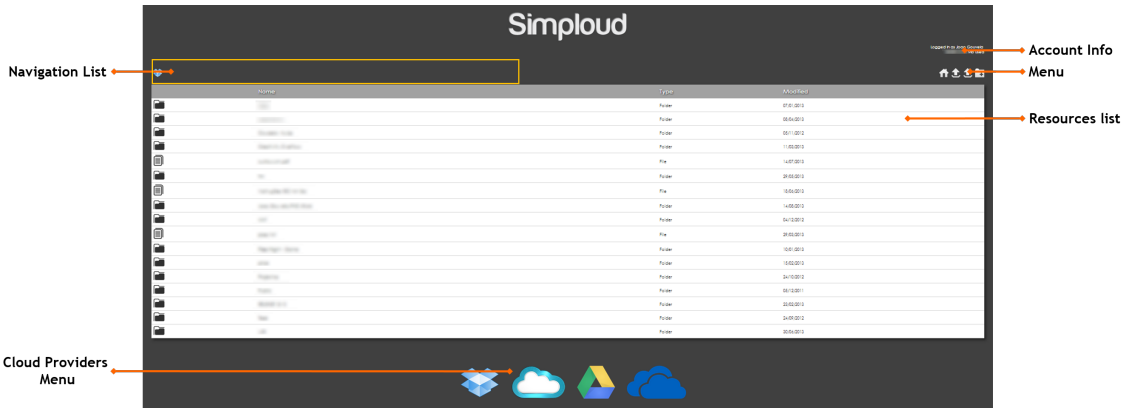

Figura 4.3: Janela que permite a visualização sobre os recursos protegidos, após autorização completada

mover e remover. Também neste caso, e pelas mesmas razões, a função de partilha não é suportada.

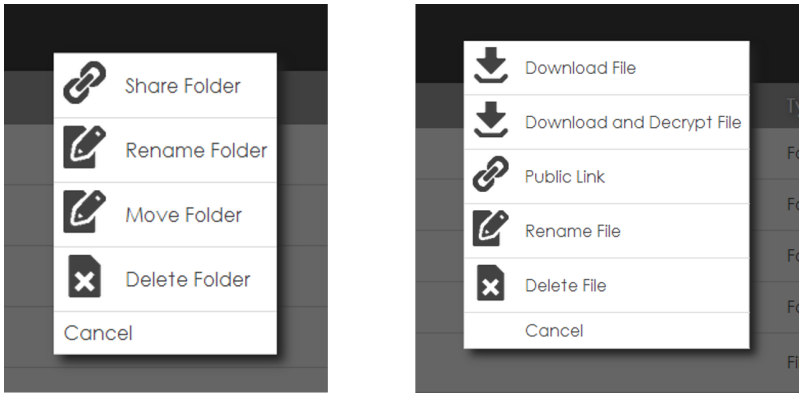

Figura 4.4: *Popups* que contêm as funções de gestão de pastas (à esquerda) e de ficheiros (à direita)

A Figura 4.5 demonstra ainda o processo de envio de ficheiros: sem recorrer à cifra (à esquerda); cifrado (à direita). Se o utilizador pretende simplesmente enviar um ficheiro, sem utilizar qualquer mecanismo de cifra, então basta selecionar o ícone sem cadeado que abre automaticamente a *popup* apresentada do lado esquerdo da Figura 4.5. Nesta janela basta indicar a localização do ficheiro em causa e clicar em *OK* para que o processo de envio inicie. Já no caso do ficheiro em causa ser motivo para a utilização dos mecanismos de cifra implementados no *Simploud*, o utilizador tem que clicar no ícone de envio que contém um cadeado e, imediatamente, é aberta a janela exemplificada à direita na Figura 4.5. Neste caso, são dadas algumas opções ao utilizador:

- Indicar a localização do ficheiro que pretende enviar cifrado;
- Ativar ou não a introdução de mecanismos de políticas de acesso. A não ativação deste mecanismo indica que o ficheiro será apenas cifrado com base na identidade do utilizador, enquanto que a opção inversa permite, para além disto, introduzir algumas políticas que controlem o processo de decifra deste. Essas políticas incluem o intervalo de tempo em que o ficheiro estará disponível para cifrar e o intervalo de *IPs* que poderão decifrar o respetivo ficheiro;
- Carregar um ficheiro de políticas existente no computador, desde que para isso cumpra e

contenha políticas implementadas neste trabalho. O não cumprimentos deste passo faz com que o ficheiro cifrado com essas políticas não volte a poder ser decifrado.

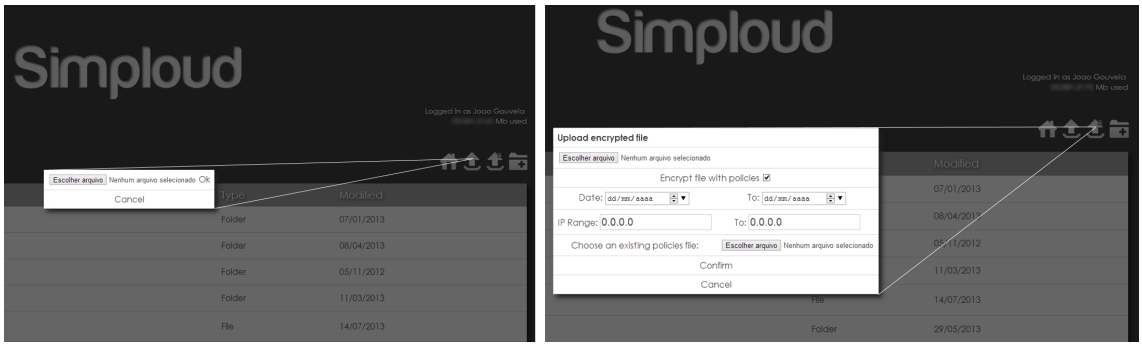

Figura 4.5: *Popups* para envio de ficheiros sem serem cifrados (à esquerda) e recorrendo a cifra (à direita)

Ao selecionar todas as opções pretendidas, basta ao utilizador clicar em *Confirm* para que o ficheiro seja cifrado e guardado com a extensão *.ibc*, ou *ibz* no caso de conter políticas.

4.2 Servidor *OpenID* proprietário

A criação de um servidor *OpenID* foi essencial para a disponibilização de um serviço que permita autenticar um utilizador titular de um cartão de identificação eletrónica em diferentes entidades que suportem este mecanismo. Neste sentido, foram desenvolvidas duas aplicações que implementam toda a estrutura *OpenID*, desde o processo de autenticação no provedor *OpenID* até ao acesso a dados privados de entidades que confiam nestes servidores de autenticação.

Para o sistema em causa é, primeiro que tudo, necessário referir que foi utilizada a biblioteca *DotNetOpenAuth* e dois dos seus projetos exemplo que implementam as duas entidades base no qual o protocolo está assente: o projeto *OpenIdProviderMvc*, que implementa um provedor de autenticação *OpenID*, e *OpenIdRelyingPartyMvc*, que se refere à entidade que confia a delegação de acesso.

O mecanismo *OpenID* pode ser iniciado em ambas as aplicações, existindo a possibilidade do utilizador se autenticar no início do processo ou apenas quando a *Relying Party* delega este processo ao servidor. Vamos considerar que um utilizador utiliza o primeiro cenário. Neste caso, o primeiro passo consiste na realização do processo de autenticação do utilizador no provedor *OpenID*. Para isso basta ao utilizador selecionar uma das opções existentes na caixa de autenticação (*Login* ou *Register*) existente na janela principal (Figura 4.6).

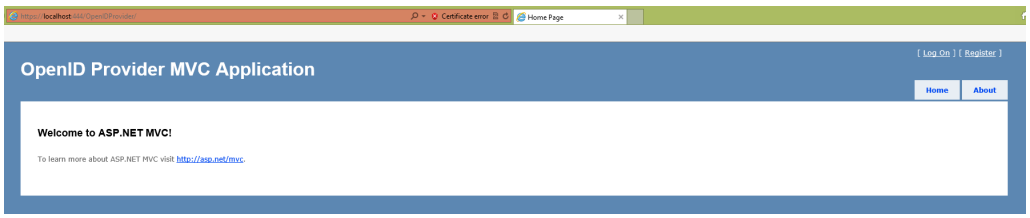

Figura 4.6: Janela inicial do provedor *OpenID*

Se estivermos perante a primeira utilização da aplicação, mesmo que o utilizador clique em *Login* será redirecionado para a página de registo, onde o primeiro procedimento da aplicação será executar o pedido de certificado do cliente (Figura 4.7) e, assim que o processo de validação do certificado esteja concluído, ligar o nome associado ao certificado apresentado juntamente com um email para encerrar o processo de registo do utilizador (Figura 4.8).

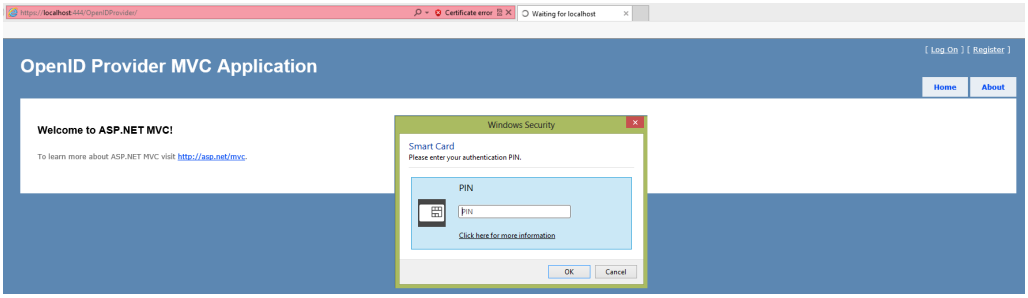

Figura 4.7: Autenticação do utilizador perante o provedor *OpenID*

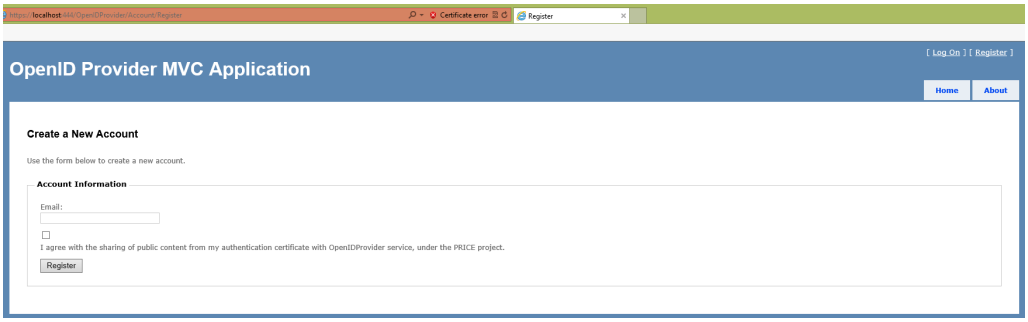

Figura 4.8: Registo do utilizador na base de dados do provedor *OpenID*

Ao concluir o processo de autenticação e registo, focamos agora o comportamento do utilizador na aplicação que irá delegar o acesso ao provedor *OpenID* (Figura 4.9). Imaginemos esta situação como um utilizador que abre uma aplicação Web e pretende autenticar-se através do seu identificador *OpenID*. É este procedimento que estamos prestes a completar.

Figura 4.9: Janela principal da aplicação que delega a autenticação

Ao entrar na área restrita, onde apenas membros autenticados pela aplicação poderão ver os conteúdos desta secção, o utilizador depara-se com uma caixa para introdução do seu identificador *OpenID* (Figura 4.10). A estrutura base deste identificador deverá seguir o seguinte formato: endereço do provedor de autenticação que gerou este identificador seguido do nome de utilizador guardado no provedor. Este identificador gera um pedido de delegação de autenticação, por parte da aplicação Web que se está a tentar aceder, ao provedor *OpenID*.

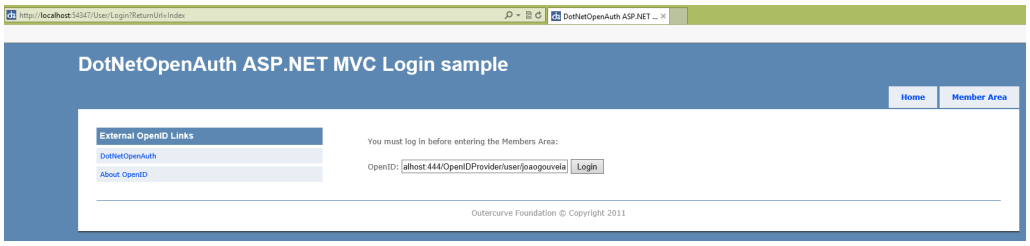

Figura 4.10: Janela restrita a membros autenticados na aplicação que delega a autenticação

Quando este recebe o pedido verifica, inicialmente, se o nome de utilizador existe e se este se encontra autenticado. Caso não esteja, o utilizador terá de se autenticar seguindo o primeiro passo, referido anteriormente. Posto isto, é mostrado ao utilizador uma janela idêntica à da Figura 4.11, onde o provedor pergunta ao utilizador se é verdade que este se está a tentar autenticar na aplicação Web que delegou este processo.

Figura 4.11: Janela do provedor *OpenID* que permite autenticar o utilizador na aplicação que confia a autenticação

Caso o utilizador confirme a tentativa de acesso na aplicação, o provedor redireciona o utilizador de novo para *Relying Party* e, daqui em diante, tem acesso à área de acesso restrito a membros da aplicação (Figura 4.12), o que significa que o utilizador se encontra autenticado nesta.

Figura 4.12: Janela restrita a membros autenticados na aplicação que delega a autenticação, com este processo concluído

4.3 Testes e tempos de execução

Na Figura 4.13 é possível observar os tempos relativos aos processos de autenticação e autorização, na aplicação *Simploud*, e os tempos de autenticação nos *Websites* dos próprios fornecedores. Para efetuar este estudo considerou-se suficiente uma amostra de vinte cinco unidades, sendo que em cada teste foi pedido:

- que o utilizador se autenticasse perante a aplicação *Simploud* e de seguida que completasse o processo de autorização. O intervalo de tempo considerado para esta situação vai

desde que o utilizador inicia o processo de autenticação, até ao momento exatamente anterior da visualização da página com os conteúdos;

- a repetição do processo anterior, mas agora sem a necessidade de proceder à autorização;
- que fizesse a sua autenticação em cada um dos três fornecedores considerados para esta arquitetura.

Ao analisar o gráfico em concreto, podemos observar que o processo de autenticação, indicado pelas barras a azul mais claro, não corresponde nem a metade do tempo de execução global, referente ao processo que se inicia com a autenticação e é concluído com o acesso aos dados protegidos. Na verdade, e embora o utilizador dezasseis tenha obtido o pior tempo de autenticação do total da amostra (15.08 segundos), a média relativa a este processo indica um valor de 10.74 segundos, o que pode ser considerado como um terço do tempo de execução da aplicação até à obtenção dos recursos privados, sabendo que a média do tempo de execução do processo de autorização é de 19.46 segundos e a média do tempo gasto nestes dois processos é igual a 30.20 segundos. O que se pretende justificar numa primeira fase com esta estatística é, considerando a principal inovação deste sistema em relação a outros existentes a introdução da autenticação via cartões de identificação eletrónica, este processo apenas requere um terço de todo o tempo de execução, o que prova que a introdução deste *token* como mecanismo de autenticação não é assim tão descabida e desproporcionada à realidade. O reforço desta análise pode ser feito através dos dados obtidos no segundo parâmetro dos testes efetuados. A principal diferença deste teste para o primeiro reside no facto de não ser necessário iniciar, de novo, o protocolo de autorização para obtenção de acesso, visto que com o primeiro teste efetuado é automaticamente guardado o *token* de acesso obtido, para futuras utilizações. Esta situação reduz drasticamente o tempo de execução da aplicação até à visualização dos recursos protegidos, sendo que a média de tempos obtidas foi de 15.55 segundos, reduzindo o tempo de execução total praticamente para metade. A linha laranja traçada ao longo do gráfico (Figura 4.13) demonstra exatamente os tempos obtidos, nesta segunda autenticação, para cada teste efetuado.

Para uma análise comparativa com os fornecedores de armazenamento na Nuvem considerados, foi pedido ao utilizador que realizasse um último teste, aceder aos seus dados protegidos através do próprio *Website* do fornecedor. Para isto, foi necessário utilizar uma ferramenta intitulada *Fiddler*, que permite a monitorização de tráfego, seja pelo protocolo *HTTP* ou *HTTPs*. Para conseguir ler o tráfego em sessões *SSL* é necessário que este programa instale o seu próprio certificado na máquina local e estabeleça algo reconhecido na área da segurança como ataque *Man-in-the-Middle*. Concluída esta breve explicação sobre a ferramenta vamos explicar qual a necessidade do uso desta. O facto de se pretender retirar informação acerca dos tempos de execução do processo de autenticação nos fornecedores considerados não é assim tão simples. Enquanto que nos primeiros dois testes a aplicação é propriedade desta dissertação, no caso dos fornecedores isso já não acontece e existe uma maneira simples de conseguir obter tempos de execução entre o começo do processo de autenticação e o acesso aos dados. O *Fiddler* facilita esta tarefa, como analisador de tráfego *Web*, permitindo a verificação de todos os parâmetros relativos aos pedidos efetuados às diferentes aplicações e serviços *Web*, incluindo a hora exata a que foram feitos esses pedidos. É neste parâmetro que se conseguem obter os tempos de execução para este terceiro teste.

O terceiro teste é relativamente simples: é pedido ao utilizador que aceda ao *Website* de cada um dos fornecedores considerados e complete o processo de autenticação em cada um deles até que lhes seja mostrada a página com os seus recursos. O *Fiddler* capta todo o tráfego gerado e bastou efetuar a análise dos pedidos feitos aos fornecedores para se conseguir obter valores temporais fidedignos. Assim, podemos indicar que a média do tempo de execução entre o ato de *login* até ao acesso aos recursos é de 19.24 segundos. A linha amarela traçada no gráfico da Figura 4.13 é a representação visual do terceiro teste efetuado.

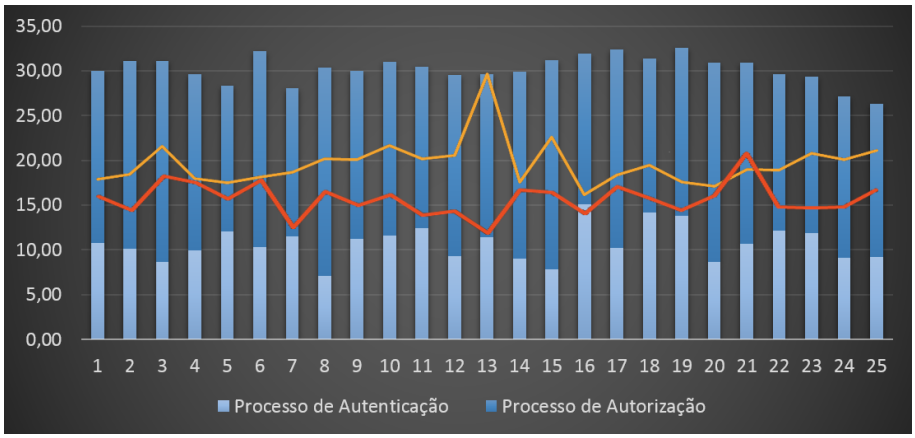

Figura 4.13: Gráfico que demonstra a relação dos tempos de execução da aplicação *Simploud* e os tempos médios de execução do formulário de Login dos fornecedores de armazenamento na Nuvem

Em relação à segunda arquitetura proposta nesta dissertação, os testes efetuados em cima servem, em parte, para justificar a utilização de cartões de identificação eletrónica como mecanismos de autenticação. Não houve a necessidade de criar mais nenhum teste para a arquitetura *OpenID* proprietária pois a única componente aglomerada aos serviços *OpenID* existentes foi mesmo a possibilidade de se efetuar autenticação com cartões *Eid*. Os resultados de tempos de execução obtidos para o primeiro teste efetuado à aplicação *Simploud* aplicam-se igualmente a este mecanismo, pois ambas as aplicações encontram-se instaladas no mesmo servidor e partilham os mesmos mecanismos de autenticação e validação do certificado apresentado.

Olhando para os testes anteriores percebe-se que com a existência de uma base de dados que guarda os *tokens* de acesso do mecanismo de autorização, se conseguem obter tempos de execução inferiores, em média, na aplicação *Simploud*. Como se pode analisar através da Figura 4.13, apenas em um caso, o tempo de execução no *Website* dos próprios fornecedores é inferior aos tempos obtidos com o uso de cartões de identificação eletrónica na aplicação *Simploud*.

Capítulo 5

Conclusão e Trabalho Futuro

A resposta dos sistemas de gestão de identidade ao problema da propagação de múltiplas identidades para o mesmo utilizador veio tornar possível a autenticação em serviços de terceiros, sem que para isso seja necessária a criação de uma nova identidade para esse mesmo serviço. Também os mecanismos de autorização existentes permitem a partilha de dados entre diferentes serviços, sem a necessidade de existir troca dos dados de autenticação (como o nome de utilizador e palavra-passe). Desta forma, caminhamos rapidamente para que no futuro, tenhamos um sistema gestor de identidades eletrónicas globalizado e que seja suportado pela maior parte dos serviços Web existentes. O que advém desta potencial conjetura é a existência de identidades digitais centralizadas que sirvam de método de autenticação em qualquer serviço existente na *Web*.

Com a evolução da internet para a computação na Nuvem, o mecanismo de autorização *OAuth* encontra-se numa posição privilegiada e tem todos os argumentos e funcionalidades para se tornar um pilar desse tipo de sistema, nomeadamente na autorização da partilha de recursos protegidos.

O trabalho elaborado e descrito nesta dissertação mostra como é possível combinar a autenticação forte de *smartcards* com a autenticação em serviços que sejam suportados pelo protocolo *OAuth* e/ou *OpenID*. O inovador sistema proposto, *Simploud*, trata uma aplicação que gere acesso uniforme a diferentes fornecedores de armazenamento na Nuvem e está assente em fortes medidas de segurança:

- criação de uma sessão *SSL* entre a aplicação e cliente (através do certificado de servidor). O protocolo de autorização *OAuth* e a entidade emissora do CC assim o exigem (Secção 3.1);
- criação de uma cadeia de certificação para gestão de todos os futuros servidores associados a estes serviços (Secção 3.1.1.1).
- implementação de autenticação através do Cartão de Cidadão (Secção 3.1.2);
- implementação do protocolo *OAuth* para comunicação com os fornecedores de armazenamento;
- implementação de mecanismos que garantam maior segurança a nível da Confidencialidade (Secção 3.2.2.2).

Para além das medidas de segurança criadas para esta primeira arquitetura foi ainda desenvolvida uma *API* que integra todo o mecanismo do protocolo *OAuth* e implementa as funções para gestão de conteúdos guardados em fornecedores *Cloud*. Esta *API*, como referido anteriormente, tem a particularidade de ser aberta (*open source*) e encontra-se já publicada na comunidade *Github*.

Da mesma forma, o segundo sistema proposto neste trabalho alia a autenticação forte do Cartão de Cidadão ao protocolo de autenticação *OpenID*. Na prática esta situação reforça o conceito anterior, tornando-a igualmente inovadora, onde a autenticação do CC serve de apoio a um protocolo que permite autorizar um consumidor a ter acesso a dados protegidos. Enquanto que nesta implementação é necessário haver uma aplicação que estabelece a ponte entre o protocolo de autorização e a autenticação via CC, existindo sempre a obrigação do consumidor e do proprietário de recursos se autenticarem, no caso da segunda implementação o mecanismo em causa permite a delegação de autenticação. Isto significa que um utilizador autenticado por um provedor *OpenID* tem acesso a qualquer aplicação Web que permita a delegação de acesso.

Existe ainda um longo caminho a percorrer, e como trabalho futuro seria importante construir uma solução de armazenamento na *Nuvem* proprietária em que a autenticação no sistema seria feita através do Cartão de Cidadão aliado ao protocolo de autenticação *OpenID*. Embora o protocolo *OAuth* não tenha sido criado para trabalhar diretamente com autenticação, este mecanismo é implementado em vários sistemas para esse efeito. Neste cenário, a implementação de um servidor de autorização *OAuth* não seria descabida, embora para esse efeito o protocolo *OpenID* seja mais eficaz. Com esta solução não haveria a necessidade de existirem aplicações de terceiros para fazerem este trabalho, como a especificada na Secção 3.2.

Um mecanismo que poderia, até certo ponto, substituir a presença de um certificado digital como *Token* de autenticação seria, por exemplo, a leitura de dados biométricos. Atualmente, a grande maioria dos cartões existentes possuem dados biométricos guardados no interior do seu *chip*. Sabendo que estes dados são únicos para cada indivíduo, seria correto implementar num servidor *OpenID* esta metodologia. Na prática, em vez da apresentação de um certificado de cliente, o utilizador teria em sua posse um leitor ótico para leitura de dados biométricos e, caso houvesse correspondência dos dados guardados no cartão este seria autenticado. As sessões *SSL* poderiam ser mantidas, visto que mesmo nas duas implementações desta dissertação, este tipo de sessão é criado com recurso aos certificados de servidor. Se aliarmos estas duas componentes dos cartões *Eid* e criarmos um mecanismo de autenticação multi-fator, na prática só se estaria a aumentar o nível de segurança deste tipo de sistemas.

Se olharmos para a conjetura atual conseguimos detetar um crescente número de dispositivos móveis no mercado e, de certa forma, podemos olhar para estes aparelhos como o futuro mais próximo da computação e do nosso quotidiano. Desta forma, e ao analisar o trabalho desenvolvido percebe-se que é impossível englobar estes dispositivos nestes mecanismos, pois ainda não existe uma maneira para que estes consigam ler cartões *Eid* direta ou indiretamente para obtenção dos certificados digitais para autenticação. Desta forma, seria igualmente interessante criar uma plataforma que permitisse interagir com estes dispositivos e que lhes fosse permitido executar implementações como as descritas neste trabalho.

Como se pode ver existe muito trabalho feito, mas há ainda muitos problemas no que toca ao tema da gestão de identidades, segurança de utilizadores, partilha de recursos ou até mesmo na privacidade dos dados protegidos. O mundo está em constante mudança, a evolução é óbvia e existirão sempre falhas e problemas que necessitem ser reparados. No término deste trabalho crê-se que houve uma contribuição notória para a atualidade do mundo informático, onde foram criadas novas soluções para autenticar utilizadores em diferentes serviços, nomeadamente em fornecedores de armazenamento na Nuvem.

Bibliografia

[AS11a] Haitham S. Al-Sinani. *Browser Extension-based Interoperation Between OAuth and Information Card-based Systems*. Technical Report Series. Mathematics Department, Royal Holloway, sep 2011. 22

[AS11b] Haitham S. Al-Sinani. Integrating oauth with information card systems. In *IAS*, pages 198–203, 2011. 22

[Bra05] Brad Fitzpatrick. Distributed identity: Yadis [online]. 2005. Available from: `http://lj-dev.livejournal.com/683939.html/`. 10

[CDM12] Giuseppina Cretella and Beniamino Di Martino. Semantic web annotation and representation of cloud apis. In *Proceedings of the 2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, EIDWT '12, pages 31–37, Washington, DC, USA, 2012. IEEE Computer Society. Available from: `http://dx.doi.org/10.1109/EIDWT.2012.61`. 23

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. Available from: `http://www.ietf.org/rfc/rfc5280.txt`. 19

[Ed.10] Ed. E. Hammer-Lahav. The oauth 1.0 protocol [online]. 2010. Available from: `http://tools.ietf.org/html/rfc5849` [cited April 2010]. 8

[Ed.12] Ed. D. Hardt. The oauth 2.0 authorization framework [online]. 2012. Available from: `http://tools.ietf.org/html/rfc6749/` [cited October 2012]. 9

[et 09a] et al. Eran Hammer-Lahav. Oauth core 1.0 revision a [online]. 2009. Available from: `http://oauth.net/core/1.0a/` [cited June 2009]. 9

[et 09b] et al. Eran Hammer-Lahav. Oauth security advisory: 2009.1 [online]. 2009. Available from: `http://oauth.net/advisories/2009-1/` [cited April 2009]. 9

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887. 20

[GCdSA13] J. Gouveia, P. Crocker, S. de Sousa, and R. Azevedo. E-id authentication and uniform access to cloud storage service providers. In *Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, CLOUDCOM '13, 2013. 3

[GCZ07] Helder Gomes, João Paulo Cunha, and André Zúquete. Authentication architecture for ehealth professionals. In *Proceedings of the 2007 OTM confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part II*, OTM'07, pages 1583–1600, Berlin, Heidelberg, 2007. Springer-Verlag. Available from: `http://dl.acm.org/citation.cfm?id=1784707.1784752`. 22

[Int08] International Civil Aviation Organization. Document 9303 [online]. 2008. 3rd Edition. Available from: http://www.icao.int/Security/mrtd/Pages/Document9303.aspx. 13

[Lei12] Barry Leiba. Oauth web authorization protocol. *IEEE Internet Computing*, 16(1):74–77, January 2012. Available from: http://dx.doi.org/10.1109/MIC.2012.11. 8

[LTHM12] E. Lear, H. Tschofenig, and H. H. Mauldin. A Simple Authentication and Security Layer (SASL) and Generic Security Service Application Program Interface (GSS-API) Mechanism for OpenID. RFC 6616 (Proposed Standard), May 2012. Available from: http://www.ietf.org/rfc/rfc6616.txt. 22

[M. 13] M. McGloin, P. Hunt, T. Lodderstedt. Oauth 2.0 threat model and security considerations [online]. 2013. Available from: http://tools.ietf.org/html/rfc6819 [cited January 2013]. 9

[MAM+99] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. Internet X.509 Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999. Available from: http://www.ietf.org/rfc/rfc2560.txt. 19

[OAS04] OASIS. Technical overview of the oasis security assertion markup language(saml) v1.1 [online]. 2004. Committee Draft. Available from: https://www.oasis-open.org/committees/download.php/6837/sstc-saml-tech-overview-1.1-cd.pdf [cited May 2004]. 8

[OAS05] OASIS. Saml v2.0 executive overview [online]. 2005. Committee Draft 01. Available from: https://www.oasis-open.org/committees/download.php/13525/sstc-saml-exec-overview-2.0-cd-01-2col.pdf [cited April 2005]. 7, 8

[OAS06] OASIS. Extensible resource identifier (xri) resolution v2.0 [online]. 2006. Working Draft 10. Available from: https://www.oasis-open.org/committees/download.php/17293 [cited March 2006]. 10

[OAS10] OASIS. Extensible resource descriptor (xrd) [online]. 2010. Version 1.0. Available from: http://docs.oasis-open.org/xri/xrd/v1.0/xrd-1.0.pdf [cited November 2010]. 10

[Ope07] OpenID Foundation. Openid authentication 2.0 - final [online]. 2007. Available from: http://openid.net/specs/openid-authentication-2_0.html [cited December 2007]. 10

[PTP10] Frank Pimenta, Claudio Teixeira, and Joaquim Sousa Pinto. Globalid - privacy concerns on a federated identity provider associated with the users' national citizen's card. In *Proceedings of the 2010 Third International Conference on Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services*, CENTRIC '10, pages 16–21, Washington, DC, USA, 2010. IEEE Computer Society. Available from: http://dx.doi.org/10.1109/CENTRIC.2010.26. 22

[RR06] David Recordon and Drummond Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM. Available from: http://doi.acm.org/10.1145/1179529.1179532. 10

[RRM12] Laurie Rae, David Recordon, and Chris Messina. *OpenID: the Definitive Guide*. Oreilly & Associates Inc, 1st edition, 2012. 42

[Sch99] Bruce Schneier. Attack Trees: Modeling Security Threats. *Dr. Dobb's Journal*, December 1999. 38

[Uri10] Pascal Urien. An openid provider based on ssl smart cards. In *Proceedings of the 7th IEEE conference on Consumer communications and networking conference*, CCNC'10, pages 444–445, Piscataway, NJ, USA, 2010. IEEE Press. Available from: `http://dl.acm.org/citation.cfm?id=1834217.1834318`. 21

[Uri11] Pascal Urien. Convergent identity: Seamless OPENID services for 3G dongles using SSL enabled USIM smart cards. In *CCNC IEEE Consumer Communications and Networking Conference*, 2011. 22

[VDF+02] W. Vesely, J. Dugan, J. Fragola, Minarick, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Handbook, National Aeronautics and Space Administration, Washington, DC, 2002. 38

[VVE10] Toby Velte, Anthony Velte, and Robert Elsenpeter. *Cloud Computing, A Practical Approach*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2010. 23

Apêndice A

Anexos

A.1 Classe *SecureController.cs*

A.1.1 Código para autenticação e validação de certificado do Cartão de Cidadão

```
var x = Request.ClientCertificate;

if (!x.IsPresent)
{
   ViewBag.Error = "No certificate selected :(";
   [...]
}

MvcApplication.SessionCertificate = new X509Certificate2(x.Certificate);

if (!AuxiliarFunctions.isCitizenCardCA(MvcApplication.SessionCertificate))
{
   ViewBag.Error = "Wrong certificate :(";
   [...]
}

X509Chain chain = new X509Chain();

X509Store store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
store.Open(OpenFlags.OpenExistingOnly | OpenFlags.ReadOnly);

foreach (X509Certificate2 mCert in store.Certificates)
{
   chain.ChainPolicy.ExtraStore.Add(mCert);
}

chain.ChainPolicy.RevocationFlag = X509RevocationFlag.EntireChain;
chain.ChainPolicy.RevocationMode = X509RevocationMode.Online;
chain.ChainPolicy.UrlRetrievalTimeout = new TimeSpan(0, 1, 0);
chain.ChainPolicy.VerificationFlags = X509VerificationFlags.NoFlag;

if (!chain.Build(MvcApplication.SessionCertificate))
{
   ViewBag.Error = "Certificate is invalid or revoked :(";
   [...]
}
store.Close();
```

A.1.2 Função que executa o pedido de autorização *OAuth* à API

```
public ActionResult AuthorizeDropbox()
{
   [...]
   var action = Url.Action("Authorized", "Secure", null, Request.Url.Scheme);
   var callbackUri = new Uri(action);

   MvcApplication.OAuthProvider = new OAuthDropbox(DropboxConsumerKey, DropboxConsumerSecret, ↩
        action);
   var requestToken = MvcApplication.OAuthProvider.GetRequestToken();

   var authorizeUri = MvcApplication.OAuthProvider.GetAuthorizeUri(requestToken);
   return Redirect(authorizeUri.ToString());
```

```
    [...]
}

public ActionResult AuthorizeCloudPT()
{
    [...]
    var action = Url.Action("Authorized", "Secure", null, Request.Url.Scheme);

    MvcApplication.OAuthProvider = new OAuthCloudpt(CloudPTConsumerKey,CloudPTConsumerSecret,action)↩
        ;
    var requestToken = MvcApplication.OAuthProvider.GetRequestToken();

    var authorizeUri = MvcApplication.OAuthProvider.GetAuthorizeUri(requestToken);
    return Redirect(authorizeUri.ToString());
    [...]
}

public ActionResult AuthorizeSkydrive()
{
    [...]
    var action = Url.Action("Authorized", "Secure", null, Request.Url.Scheme);
    action = Url.Encode(action);

    MvcApplication.OAuthProvider = new OAuthSkydrive(SkydriveConsumerKey, SkydriveConsumerSecret, ↩
        action);

    var uri = MvcApplication.OAuthProvider.GetAuthorizeUri(null);
    return Redirect(uri.ToString());
}
```

A.2 Classe *AuxiliarFunctions.cs*

A.2.1 Função para verificar se o certificado apresentado pertence aos certificados do Cartão de Cidadão

```
public static bool isCitizenCardCA(X509Certificate2 cert)
{
    if (cert.Issuer == "CN=EC de Autenticação do Cartão de Cidadão 0001, OU=subECEstado, O=Cartão de↩
        Cidadão, C=PT" ||
       cert.Issuer == "CN=EC de Autenticação do Cartão de Cidadão 0002, OU=subECEstado, O=Cartão de ↩
           Cidadão, C=PT" ||
       cert.Issuer == "CN=EC de Autenticação do Cartão de Cidadão 0003, OU=subECEstado, O=Cartão de ↩
           Cidadão, C=PT" ||
       cert.Issuer == "CN=EC de Autenticação do Cartão de Cidadão 0004, OU=subECEstado, O=Cartão de ↩
           Cidadão, C=PT" ||
       cert.Issuer == "CN=EC de Autenticação do Cartão de Cidadão 0005, OU=subECEstado, O=Cartão de ↩
           Cidadão, C=PT" ||
       cert.Issuer == "CN=EC de Autenticação do Cartão de Cidadão 0006, OU=subECEstado, O=Cartão de ↩
           Cidadão, C=PT" ||
       cert.Issuer == "CN=EC de Autenticação do Cartão de Cidadão 0007, OU=subECEstado, O=Cartão de ↩
           Cidadão, C=PT" ||
       cert.Issuer == "CN=EC de Autenticação do Cartão de Cidadão 0008, OU=subECEstado, O=Cartão de ↩
           Cidadão, C=PT")
       return true;
    else
       return false;
}
```

A.2.2 Função que executa o pedido à API para obtenção de *Token* de acesso

```
public static OAuthToken requestAccessToken(OAuthToken requestToken, string verification, string ↩
    access_token, int cloud_ID, out string error)
{
    [...]
    var accessToken = MvcApplication.OAuthProvider.GetAccessToken(requestToken, verification);
    [...]
    return accessToken;
```

```
}

public static string refreshToken(UrlHelper url, int provider)
{
   [...]
   dbo.executeNonQuery("DELETE FROM Register WHERE IssuerName='" + cert.IssuerName + "' AND ↩
        SerialNumber='" + cert.SerialNumber + "' AND Cloud_ID=" + provider +";");
   uri = url.Action("AuthorizeDropbox", "Secure");
   return uri;
}
```

A.3 Classes pertencentes à API desenvolvida

A.3.1 Classe *OAuthBaseCloudpt.cs*

```
ï»¿internal class OAuthBaseCloudpt : IOAuthBase
{
   protected class QueryParameter
   {
      [...]
   }

   protected class QueryParameterComparer : IComparer<QueryParameter>
   {
      [...]
   }

   [...] //List of know and used oauth parameters' names
   protected const string OAuthVerifier = "oauth_verifier";

   private string ComputeHash(HashAlgorithm hashAlgorithm, string data)
   {
      [...]
   }

   private List<QueryParameter> GetQueryParameters(string parameters)
   {
      [...]
   }

   public string UrlEncode(string value)
   {
      [...]
   }

   protected string NormalizeRequestParameters(IList<QueryParameter> parameters)
   {
      [...]
   }

   public string GenerateSignatureBase(Uri url, string consumerKey, string token, string ↩
        tokenSecret, string httpMethod, string timeStamp, string nonce, string signatureType, out ↩
        string normalizedUrl, out string normalizedRequestParameters, string callback, string ↩
        verification)
   {
      [...]

      if(verification == null)
      {
         verification = string.Empty;
      }

      [...]

      if (url.AbsolutePath.Contains("request_token") || url.AbsolutePath.Contains("authorize"))
      {
         if (!string.IsNullOrEmpty(callback))
            parameters.Add(new QueryParameter(OAuthCallbackKey, callback));
```

```
            else
               parameters.Add(new QueryParameter(OAuthCallbackKey, "oob"));
         }

         if (!string.IsNullOrEmpty(token))
            parameters.Add(new QueryParameter(OAuthTokenKey, token));

         if (!string.IsNullOrEmpty(verification))
            parameters.Add(new QueryParameter(OAuthVerifier, verification));

         [...]
      }

      public string GenerateSignatureUsingHash(string signatureBase, HashAlgorithm hash)
      {
         [...]
      }

      public string GenerateSignature(Uri url, string consumerKey, string consumerSecret, string token↩
           , string tokenSecret, string httpMethod, string timeStamp, string nonce, out string ↩
           normalizedUrl, out string normalizedRequestParameters, string callback, string verification↩
           )
      {
         [...]
      }

      public string GenerateSignature(Uri url, string consumerKey, string consumerSecret, string token↩
           , string tokenSecret, string httpMethod, string timeStamp, string nonce, Simploud.Api.OAuth↩
           .SignatureTypes signatureType, out string normalizedUrl, out string ↩
           normalizedRequestParameters, string callback, string verification)
      {
         [...]
      }

      public virtual string GenerateTimeStamp()
      {
         [...]
      }

      public virtual string GenerateNonce()
      {
         [...]
      }

}
```

A.3.2 Classe *OAuthDropbox.cs*

```
public class OAuthDropbox : IOAuth
{
   private readonly IOAuthBase _oAuthBase;
   private const string DropboxApiVersion = "1";
   private const string DropboxBaseUri = "https://api.dropbox.com/" + DropboxApiVersion + "/";
   private const string DropboxAuthorizeBaseUri = "https://www.dropbox.com/" + DropboxApiVersion + ↩
        "/";
   private string _consumerKey, _consumerSecret, _callbackUrl;

   public OAuthDropbox(string consumerKey, string consumerSecret, string callbackUrl = null)
   {
      _oAuthBase = new OAuthBaseDropbox();
      _consumerKey = consumerKey;
      _consumerSecret = consumerSecret;
      _callbackUrl = callbackUrl;
   }

   public OAuthToken GetRequestToken()
   {
      var uri = new Uri(new Uri(DropboxBaseUri), "oauth/request_token");
      if(_callbackUrl != null)
```

```
        _callbackUrl = _oAuthBase.UrlEncode(_callbackUrl);
      uri = SignRequest(uri, _consumerKey, _consumerSecret, _callbackUrl);
      var request = (HttpWebRequest) WebRequest.Create(uri);
      request.Method = WebRequestMethods.Http.Get;
      var response = request.GetResponse();
      var queryString = new StreamReader(response.GetResponseStream()).ReadToEnd();
      var parts = queryString.Split('&');
      var token = parts[1].Substring(parts[1].IndexOf('=') + 1);
      var secret = parts[0].Substring(parts[0].IndexOf('=') + 1);
      return new OAuthToken(token, secret);
   }

   public Uri GetAuthorizeUri(OAuthToken requestToken)
   {
      string query_string = null;
      query_string = String.Format("oauth_token={0}", requestToken.Token);
      if (_callbackUrl != null)
         query_string = String.Format("{0}&&oauth_callback={1}", query_string, _callbackUrl);
      var authorizeUri = String.Format("{0}{1}?{2}", new Uri(DropboxAuthorizeBaseUri), "oauth/↩
          authorize", query_string);
      return new Uri(authorizeUri);
   }

   public OAuthToken GetAccessToken(OAuthToken requestToken, string verification = null)
   {
      var uri = new Uri(new Uri(DropboxBaseUri), "oauth/access_token");
      uri = SignRequest(uri, _consumerKey, _consumerSecret, null, requestToken, verification);
      var request = (HttpWebRequest) WebRequest.Create(uri);
      request.Method = WebRequestMethods.Http.Get;
      var response = request.GetResponse();
      var reader = new StreamReader(response.GetResponseStream());
      var accessToken = reader.ReadToEnd();
      var parts = accessToken.Split('&');
      var token = parts[1].Substring(parts[1].IndexOf('=') + 1);
      var secret = parts[0].Substring(parts[0].IndexOf('=') + 1);
      return new OAuthToken(token, secret);
   }

   public Uri SignRequest(Uri uri, string consumerKey, string consumerSecret, OAuthToken token, ↩
       string httpMethod, string callbackUrl = null, string verification = null)
   {
      var nonce = _oAuthBase.GenerateNonce();
      var timestamp = _oAuthBase.GenerateTimeStamp();
      string parameters;
      string normalizedUrl;
      string callback = callbackUrl == null ? String.Empty : callbackUrl;
      string requestToken = token == null ? String.Empty : token.Token;
      string tokenSecret = token == null ? String.Empty : token.Secret;
      string verify = verification == null ? String.Empty : verification;
      var signature = _oAuthBase.GenerateSignature(
         uri, consumerKey, consumerSecret,
         requestToken, tokenSecret, httpMethod, timestamp,
         nonce, SignatureTypes.HMACSHA1,
         out normalizedUrl, out parameters, callback, verification);
      var requestUri = String.Format("{0}?{1}&oauth_signature={2}",
         normalizedUrl, parameters, _oAuthBase.UrlEncode(signature));
      return new Uri(requestUri);
   }

   public Uri SignRequest(Uri uri, string consumerKey, string consumerSecret, string callbackUrl = ↩
       null, OAuthToken token = null, string verification = null)
   {
      return SignRequest(uri, consumerKey, consumerSecret, token, "GET", callbackUrl, verification)↩
          ;
   }
}
```

A.3.3 Classe *DropboxCloudProvider.cs*

```
ï»¿public class DropboxCloudProvider : ICloudProvider
```

```
{
   private const string DropboxApiVersion = "1";
   private const string DropboxBaseUri = "https://api.dropbox.com/" + DropboxApiVersion + "/";
   private const string DropboxAuthorizeBaseUri = "https://www.dropbox.com/" + DropboxApiVersion + ↩
        "/";
   private const string DropboxApiContentServer = "https://api-content.dropbox.com/" + ↩
        DropboxApiVersion + "/";

   private readonly OAuthToken _accessToken;
   private readonly string _consumerKey;
   private readonly string _consumerSecret;
   private readonly string _root;

   public DropboxCloudProvider(string consumerKey, string consumerSecret, OAuthToken accessToken, ↩
        string root)
   {
      _accessToken = accessToken;
      _consumerKey = consumerKey;
      _consumerSecret = consumerSecret;
      _root = root;
   }

   private string GetRequest(Uri uri)
   {
      var oauth = new OAuth.OAuthDropbox(_consumerKey, _consumerSecret, null);
      var requestUri = oauth.SignRequest(uri, _consumerKey, _consumerSecret, _accessToken, "GET", ↩
           null, null);
      HttpWebRequest request = null;
      request = (HttpWebRequest)WebRequest.Create(requestUri);
      request.Method = WebRequestMethods.Http.Get;
      var response = request.GetResponse();
      var reader = new StreamReader(response.GetResponseStream());
      return reader.ReadToEnd();
   }

   private string PostRequest(Uri uri)
   {
      var oauth = new OAuth.OAuthDropbox(_consumerKey, _consumerSecret, null);
      var requestUri = oauth.SignRequest(uri, _consumerKey, _consumerSecret, _accessToken, "POST", ↩
           null, null);
      var req = requestUri.AbsoluteUri.Split('?');
      var request = (HttpWebRequest)WebRequest.Create(uri);
      request.Method = WebRequestMethods.Http.Post;
      request.ContentType = "application/x-www-form-urlencoded";
      ASCIIEncoding encoding = new ASCIIEncoding();
      byte[] data = encoding.GetBytes(req[1]);
      request.ContentLength = data.Length;
      using (Stream stream = request.GetRequestStream())
      {
         stream.Write(data, 0, data.Length);
      }
      var response = request.GetResponse();
      var reader = new StreamReader(response.GetResponseStream());
      return reader.ReadToEnd();
   }

   public Account GetAccountInfo()
   {
      Uri uri = new Uri(new Uri(DropboxBaseUri), "account/info");
      var json = GetRequest(uri);
      json = UidFix(json);
      return ParseJson<Account>(json);
   }

   public Simploud.Api.Models.Folder GetFiles(string path)
   {
      if (path != "" && path[0] == '/')
         path = path.Remove(0, 1);
      Uri uri = new Uri(new Uri(DropboxBaseUri), String.Format("metadata/{0}/{1}", _root, path));
      var json = GetRequest(uri);
```

```
    return ParseJson<Simploud.Api.Models.Folder>(json);
}

public Simploud.Api.Models.File CreateFolder(string path, string name = null)
{
    if (path == "" || path[0] != '/')
        path = "/" + path;
    if (path != "" && path[path.Length - 1] != '/')
        path = path + "/";
    Uri uri = new Uri(new Uri(DropboxBaseUri),
            String.Format("fileops/create_folder?root={0}&path={1}", _root, UpperCaseUrlEncode(↩
                 path + name)));
    var json = GetRequest(uri);
    return ParseJson<Simploud.Api.Models.File>(json);
}

public Simploud.Api.Models.File Rename(string old_name, string new_name)
{
    if (old_name != "" && old_name[0] != '/')
        old_name = "/" + old_name;
    new_name = newNamePath(old_name, new_name);
    Uri uri = new Uri(new Uri(DropboxBaseUri),
            String.Format("fileops/move?root={0}&from_path={1}&to_path={2}",
            _root, UpperCaseUrlEncode(old_name), UpperCaseUrlEncode(new_name)));
    var json = GetRequest(uri);
    return ParseJson<Simploud.Api.Models.File>(json);
}

public Simploud.Api.Models.File Move(string fromPath, string toPath)
{
    if (fromPath != "" && fromPath[0] == '/')
        fromPath = fromPath.Remove(0, 1);
    if (toPath != "" && toPath[0] == '/')
        toPath = toPath.Remove(0, 1);
    Uri uri = new Uri(new Uri(DropboxBaseUri),
            String.Format("fileops/move?root={0}&from_path={1}&to_path={2}",
            _root, UpperCaseUrlEncode(fromPath), UpperCaseUrlEncode(toPath)));
    var json = GetRequest(uri);
    return ParseJson<Simploud.Api.Models.File>(json);
}

public Simploud.Api.Models.File Delete(string path)
{
    if (path != "" && path[0] == '/')
        path = path.Remove(0, 1);
    Uri uri = new Uri(new Uri(DropboxBaseUri),
            String.Format("fileops/delete?root={0}&path={1}",
            _root, UpperCaseUrlEncode(path)));
    var json = GetRequest(uri);
    return ParseJson<Simploud.Api.Models.File>(json);
}

public byte[] DownloadFile(string path)
{
    if (path != "" && path[0] == '/')
        path = path.Remove(0, 1);
    Uri uri = new Uri(new Uri(DropboxApiContentServer),
          String.Format("files?root={0}&path={1}",
          _root, UpperCaseUrlEncode(path)));
    var oauth = new OAuth.OAuthDropbox(_consumerKey, _consumerSecret, null);
    var requestUri = oauth.SignRequest(uri, _consumerKey, _consumerSecret, null, _accessToken);
    var request = (HttpWebRequest)WebRequest.Create(requestUri);
    request.Method = WebRequestMethods.Http.Get;
    var response = request.GetResponse();
    var metadata = response.Headers["x-" + _root + "-metadata"];
    var file = ParseJson<Simploud.Api.Models.File>(metadata);
    using (Stream responseStream = response.GetResponseStream())
    using (MemoryStream memoryStream = new MemoryStream())
    {
        byte[] buffer = new byte[1024];
```

```
            int bytesRead;
            do
            {
               bytesRead = responseStream.Read(buffer, 0, buffer.Length);
               memoryStream.Write(buffer, 0, bytesRead);
            } while (bytesRead > 0);

            return memoryStream.ToArray();
         }
      }

      public Simploud.Api.Models.File UploadFile(string path, string filename, MemoryStream data)
      {
         if (path != "" && path[0] == '/')
            path = path.Remove(0, 1);
         if(path != "" && path[path.Length-1] != '/')
            path = path + "/";
         Uri uri = new Uri(new Uri(DropboxApiContentServer),
               String.Format("files_put/{0}/{1}{2}",
               _root, UpperCaseUrlEncode(path), filename));
         var oauth = new OAuth.OAuthDropbox(_consumerKey, _consumerSecret, null);
         var requestUri = oauth.SignRequest(uri, _consumerKey, _consumerSecret, _accessToken, "PUT");
         var request = (HttpWebRequest)WebRequest.Create(requestUri);
         request.Method = WebRequestMethods.Http.Put;
         request.KeepAlive = true;
         byte[] buffer = data.ToArray();
         request.ContentLength = buffer.Length;
         using (var requestStream = request.GetRequestStream())
         {
            requestStream.Write(buffer, 0, buffer.Length);
         }
         var response = (HttpWebResponse)request.GetResponse();
         var reader = new StreamReader(response.GetResponseStream());
         var json = reader.ReadToEnd();
         return ParseJson<Simploud.Api.Models.File>(json);
      }
   }
```

A.3.4 Modelos criados para desserializar as respostas JSON

```
ï»¿[JsonObject(MemberSerialization.OptIn)]
public class AccessToken
{
   [JsonProperty(PropertyName = "access_token")]
   public string access_token { get; internal set; }
   [JsonProperty(PropertyName = "expires_in")]
   public int expires_in { get; internal set; }
   [JsonProperty(PropertyName = "scope")]
   public string scope { get; internal set; }
   [JsonProperty(PropertyName = "token_type")]
   public string token_type { get; internal set; }
}

[JsonObject(MemberSerialization.OptIn)]
public class Quota
{
   [JsonProperty(PropertyName = "quota")]
   public long Total { get; internal set; }
   [JsonProperty(PropertyName = "shared")]
   public long Shared { get; internal set; }
   [JsonProperty(PropertyName = "normal")]
   public long Normal { get; internal set; }
   [JsonProperty(PropertyName = "available")]
   public long Available { get; internal set; }
}

[JsonObject(MemberSerialization.OptIn)]
public class Account
{
   [JsonProperty(PropertyName = "uid")]
```

```
    public string Id { get; internal set; }
    [JsonProperty(PropertyName = "referral_link")]
    public string ReferralLink { get; internal set; }
    [JsonProperty(PropertyName = "name")]
    public string Name { get; internal set; }
    [JsonProperty(PropertyName = "display_name")]
    public string DisplayName { get; internal set; }
    [JsonProperty(PropertyName = "email")]
    public string Email { get; internal set; }
    [JsonProperty(PropertyName = "country")]
    public string Country { get; internal set; }
    [JsonProperty(PropertyName = "quota_info")]
    public Quota Quota { get; internal set; }
}

[JsonObject(MemberSerialization.OptIn)]
public class Folder
{
    [JsonProperty(PropertyName = "size")]
    public string Size { get; internal set; }
    [JsonProperty(PropertyName = "rev")]
    public string Revision { get; internal set; }
    [JsonProperty(PropertyName = "thumb_exists")]
    public bool ThumbnailExists { get; internal set; }
    [JsonProperty(PropertyName = "bytes")]
    public long Bytes { get; internal set; }
    [JsonProperty(PropertyName = "is_dir")]
    public bool IsDirectory { get; internal set; }
    [JsonProperty(PropertyName = "root")]
    public string Root { get; internal set; }
    [JsonProperty(PropertyName = "icon")]
    public string Icon { get; internal set; }
    [JsonProperty(PropertyName = "mime_type")]
    public string MimeType { get; internal set; }
    [JsonProperty(PropertyName = "path")]
    public string Path { get; internal set; }
    [JsonProperty(PropertyName = "contents")]
    public IEnumerable<File> Contents { get; internal set; }
    [JsonProperty(PropertyName = "modified")]
    public DateTime Modified { get; internal set; }
    //skydrive params
    [JsonProperty(PropertyName = "data")]
    public IEnumerable<File> Data { get; internal set; }
    [JsonProperty(PropertyName = "id")]
    public string Id { get; internal set; }
    [JsonProperty(PropertyName = "from")]
    public From From_data { get; internal set; }
    [JsonProperty(PropertyName = "name")]
    public string Name { get; internal set; }
    [JsonProperty(PropertyName = "description")]
    public string Description { get; internal set; }
    [JsonProperty(PropertyName = "parent_id")]
    public string ParentId { get; internal set; }
    [JsonProperty(PropertyName = "upload_location")]
    public string Upload_location { get; internal set; }
    [JsonProperty(PropertyName = "comments_count")]
    public long Comments_count { get; internal set; }
    [JsonProperty(PropertyName = "comments_enabled")]
    public bool Comments_enabled { get; internal set; }
    [JsonProperty(PropertyName = "is_embeddable")]
    public bool Is_embeddable { get; internal set; }
    [JsonProperty(PropertyName = "source")]
    public string Source { get; internal set; }
    [JsonProperty(PropertyName = "link")]
    public string Link { get; internal set; }
    [JsonProperty(PropertyName = "type")]
    public string Type { get; internal set; }
    [JsonProperty(PropertyName = "shared_with")]
    public Shared_with Shared_with { get; internal set; }
    [JsonProperty(PropertyName = "created_time")]
```

```
    public string Created_time { get; internal set; }
    [JsonProperty(PropertyName = "updated_time")]
    public string Updated_time { get; internal set; }
}

[JsonObject(MemberSerialization.OptIn)]
public class From
{
    [JsonProperty(PropertyName = "name")]
    public string Name { get; internal set; }
    [JsonProperty(PropertyName = "id")]
    public string Id { get; internal set; }
}

[JsonObject(MemberSerialization.OptIn)]
public class Shared_with
{
    [JsonProperty(PropertyName = "access")]
    public string Access { get; internal set; }
}

[JsonObject(MemberSerialization.OptIn)]
public class File
{
    [JsonProperty(PropertyName = "size")]
    public string Size { get; internal set; }
    [JsonProperty(PropertyName = "rev")]
    public string Revision { get; internal set; }
    [JsonProperty(PropertyName = "thumb_exists")]
    public bool ThumbnailExists { get; internal set; }
    [JsonProperty(PropertyName = "bytes")]
    public long Bytes { get; internal set; }
    [JsonProperty(PropertyName = "modified")]
    public DateTime Modified { get; internal set; }
    [JsonProperty(PropertyName = "path")]
    public string Path { get; internal set; }
    [JsonProperty(PropertyName = "is_dir")]
    public bool IsDirectory { get; internal set; }
    [JsonProperty(PropertyName = "icon")]
    public string Icon { get; internal set; }
    [JsonProperty(PropertyName = "root")]
    public string Root { get; internal set; }
    [JsonProperty(PropertyName = "is_deleted")]
    public bool IsDeleted { get; internal set; }
    //skydrive params
    [JsonProperty(PropertyName = "id")]
    public string Id { get; internal set; }
    [JsonProperty(PropertyName = "from")]
    public From From_data { get; internal set; }
    [JsonProperty(PropertyName = "name")]
    public string Name { get; internal set; }
    [JsonProperty(PropertyName = "description")]
    public string Description { get; internal set; }
    [JsonProperty(PropertyName = "parent_id")]
    public string ParentId { get; internal set; }
    [JsonProperty(PropertyName = "upload_location")]
    public string Upload_location { get; internal set; }
    [JsonProperty(PropertyName = "comments_count")]
    public long Comments_count { get; internal set; }
    [JsonProperty(PropertyName = "comments_enabled")]
    public bool Comments_enabled { get; internal set; }
    [JsonProperty(PropertyName = "is_embeddable")]
    public bool Is_embeddable { get; internal set; }
    [JsonProperty(PropertyName = "source")]
    public string Source { get; internal set; }
    [JsonProperty(PropertyName = "link")]
    public string Link { get; internal set; }
    [JsonProperty(PropertyName = "type")]
    public string Type { get; internal set; }
    [JsonProperty(PropertyName = "shared_with")]
```

```
   public Shared_with Shared_with { get; internal set; }
   [JsonProperty(PropertyName = "created_time")]
   public string Created_time { get; internal set; }
   [JsonProperty(PropertyName = "updated_time")]
   public string Updated_time { get; internal set;
   public byte[] Data { get; internal set; }
   public void Save(string path)
   {
      using (var fileStream = new FileStream(
         path, FileMode.Create, FileAccess.ReadWrite))
      {
         fileStream.Write(Data, 0, Data.Length);
      }
   }
}
```

Glossário

CA	A *Certification Authority* é uma entidade que, dentro de uma estrutura PKI, emite certificados digitais para diferentes fins.
Cartões Eid	Os cartões de identificação eletrónica, entre outras funções e tal como o nome indica, permite ao seu titular identificar-se digitalmente perante outras entidades.
CRL	As *Certification Revocation List* são listas criadas numa estrutura PKI e que guardam os certificados inválidos, suspensos, revogados ou expirados.
IBE	O mecanismo de *Identity Based Encryption* permite a cifra de informação baseando-se na identidade do utilizador que pretende utilizar este processo.
Idm	Um *Identity Management System* é um sistema que tem como principal função gerir a identidade dos utilizadores. Esta gestão pode ser efetuada sobre diversas formas: gestão de identidades, controlo de acesso, serviços de controlo de diretoria ou protocolos e padrões.
OAuth	Protocolo de autorização que permite a uma terceira entidade aceder a recursos protegidos num servidor privado, com a devida autorização do proprietário desses recursos e sem troca de dados de autenticação.
OCSP	Protocolo que permite obter o estado de um determinado certificado.
OpenID	Protocolo de autenticação que permite a um utilizador usar as mesmas credenciais para aceder a diferentes serviços e aplicações, tantos quantos os que implementem a delegação de autenticação imposta por este mecanismo.
PKI	Uma *Public Key Infrastructure* designa-se por todo o conjunto de *hardware*, *software*, pessoas, políticas, e procedimentos necessário para a criação, gestão, distribuição, utilização, armazenamento, e revogação de certificados digitais.
RA	A principal função da *Registration Authority* é estabelecer a ligação entre o utilizador e a CA no processo de criação de um certificado.
VA	A *Validation Authority*, em caso de existência numa estrutura PKI, fornece informação acerca na validação em vez da CA.