UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Aplicações de Criptografia Baseada em Identidade com Cartões de Identificação Eletrónica

João Pedro Cipriano Silveira

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

Orientador: Prof. Doutor Paul Andrew Crocker

Covilhã, outubro de 2013

Dedicatória

Dedico esta dissertação aos meus pais
Pelos ensinamentos e valores transmitidos
E pelo apoio contínuo e dedicação incondicional

Agradecimentos

Em primeiro lugar, gostaria de agradecer aos meus pais pelo apoio, incentivo e confiança depositada em mim ao longo do meu percurso académico bem como ao longo de toda a minha vida. Sem a dedicação e o esforço deles não teria sido possível atingir este objetivo.

Quero agradecer ao meu irmão pelos momentos de descontração e pela ajuda na revisão desta dissertação.

Quero também agradecer à minha melhor amiga e namorada, Joana Pereira, pela motivação, paciência e carinho dado durante estes dois anos de Mestrado. Mesmo longe nunca deixou de me dar forças para continuar a lutar pelos meus objetivos e dar o meu melhor em tudo o que faço.

Ao Prof. Doutor Paul Andrew Crocker estou profundamente agradecido pela oportunidade que me foi dada ao integrar o projeto ***Privacy, Reliability and Integrity in Cloud Environments*** (PRICE) e pelo seu papel como orientador. Os seus conhecimentos, observações, prontidão e tempo despendido foram de grande importância para a realização desta dissertação ao longo de todas as etapas.

Quero agradecer também ao Prof. Doutor Simão Melo de Sousa pelo seu espírito crítico e pelos seus pareceres que se tornaram indispensáveis ao longo das várias fases deste projeto.

Ao Ricardo Azevedo da ***PT Inovação*** pelo interesse e apoio dado ao longo de todo o desenvolvimento do projeto e dissertação.

Quero expressar os meus agradecimentos ao meu colega de Mestrado e amigo, João Gouveia, tanto pela ajuda, *brainstorming* e colaboração no âmbito do projeto PRICE que engloba ambas as nossas dissertações, como pelos momentos de descontração proporcionados dentro e fora do laboratório.

Gostaria de agradecer também aos meus amigos e colegas de laboratório, Leopoldo Ismael e Ruben Espírito Santo, pelo ânimo, distração e companhia nas longas noites de trabalho e café.

Por último gostaria de agradecer aos restantes colegas do ***RELiablE And SEcure Computation Group*** (RELEASE) e aos meus amigos, Cláudia Caronho, Vitor Rolo, Pedro Querido e Samuel Dias.

"Aqueles que passam por nós, não vão sós, não nos deixam sós. Deixam um pouco de si, levam um pouco de nós." - Antoine de Saint-Exupéry

Resumo

A presente dissertação resulta da investigação sobre Criptografia Baseada em Identidade, cujo o objetivo principal é a criação de um sistema que utilize os mecanismos de autenticação de um cartão de identificação eletrónica (EID Card), em particular do Cartão de Cidadão (CC) Português, e as características de identidade presentes no cartão para permitir a cifra e decifra de ficheiros. O sistema foi idealizado com o objetivo de estender as capacidades criptográficas do CC, proporcionando ao mesmo tempo um sistema seguro, inovador, simples e transparente na sua utilização e nas suas funcionalidades para o utilizador comum.

O desenvolvimento deste projeto é justificado com o aumento visível da procura de soluções que garantam a privacidade e confidencialidade de dados armazenados em serviços de informação e em particular na *Cloud*.

Neste trabalho é proposta, de forma detalhada, uma implementação de um sistema para a Privacidade e Confidencialidade de ficheiros, utilizando Criptografia Baseada em Identidade e o Cartão de Cidadão. Este sistema permite a cifra de ficheiros com múltiplas identidades (chaves públicas) e a associação de políticas de privacidade e acesso flexíveis a esses ficheiros. A cifra é efetuada sem a necessidade de partilha prévia de chaves, utilizando a identidade do utilizador como chave pública. A respetiva criação e distribuição de chaves privadas é gerida por uma terceira entidade, na qual irá depender a segurança do sistema e que necessita do mecanismo de autenticação forte do Cartão de Cidadão. O sistema final está implementado em várias aplicações para *Desktop*, como serviço faz parte de um sistema mais vasto para armazenamento em *Cloud*.

Palavras-chave

Cartão de Cidadão, Criptografia, Criptografia Baseada em Identidade, Segurança, Autenticação, Confidencialidade, Privacidade, Políticas de Ficheiros, Cloud

Abstract

The present dissertation results of the investigation on Identity Based Cryptography, having as goal the creation of a system that uses the authentication mecanisms of an Electronic ID Card (EID Card), more specifically the Portuguese EID Card, and the identity characteristics present on this card to allow the encryption and decryption of files. The sistem was designed with the purpose of extending the cryptographic abilities of the card, providing at the same time a secure, innovative, simple and transparent in its use and functionalities, system to the common user.

The development of this project is justified with the noticeable increase in demand for solutions that ensure the privacy and confidentiality of data stored in information services and particulary in *Cloud* services.

This work proposes in detail an implementation of a system for the Privacy and Confidentiality of files using Identity Based Encryption and the Portuguese EID Card. This system allows the encryption of files with multiple identities (public keys) and the association of flexible privacy and access policies. The encryption process is executed without the necessity of previous key sharing, using the users identity as the public key. The creation and distribution of private keys is managed by a third party in which will depend the security of the system and that needs the strong authentication mechanism of the EID card. The final system is deployed in various desktop applications and as a service being part of a wider system for *Cloud* storage.

Keywords

Citizen Card, Cryptography, Identity Based Cryptography, Security, Authentication, Confidentiality, Privacy, File Policies, Cloud

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AES	Advanced Encryption Standard
AIBE	Authenticated Identity-Based Encryption
API	Application Programming Interface
CA	Certification Authority
CBC	Cipher Block Chaining
CC	Cartão de Cidadão
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
CSP	Cryptographic Service Provider
DH	Diffie-Hellman
DLL	Dynamic-Link Library
EID	Electronic Identity
EMV-CAP	Europay, Mastercard and Visa Chip Authentication Program
EPAL	Enterprise Privacy Authorization Language
GUID	Globally Unique Identifier
HIBE	Hierarchical Identity-Based Encryption
IBC	Identity Based Cryptography
IBE	Identity Based Encryption
IBS	Identity Based Signature
IP	Internet Protocol
IV	Initialization Vector
NSA	National Security Agency
OCSP	Online Certificate Status Protocol
OTP	One-time Password
PIN	Personal Identification Number
PKCS	Public-Key Cryptography Standards
PKG	Private Key Generator
PKI	Public Key Infrastructure
PPS	Public Parameter Server
PUK	PIN Unlock Key
RFC	Request For Comments
RSA	Rivest, Shamir, Adelman cryptographic algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UBI	Universidade da Beira Interior
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language
XrML	eXtensible Rights Markup Language

Capítulo 1

Introdução

Com a massificação do uso da Internet, o constante crescimento de redes informáticas e de utilização de serviços em *Cloud*, surgem dúvidas quanto à privacidade e confidencialidade dos dados armazenados. A questão da segurança toma assim especial importância, especialmente se imaginarmos a possibilidade de informações confidenciais de particulares ou empresas estarem expostas a intrusos e atacantes na Internet, com meios cada vez mais sofisticados, que comprometem a segurança e a privacidade dessas informações.

Desta forma, a criptografia tem especial importância na salvaguarda da confidencialidade e privacidade dos dados, fazendo com que apenas as entidades às quais pertencem ou foram destinadas as informações possam compreendê-las. A criptografia fornece técnicas pelas quais a informação pode ser codificada e descodificada, de forma a que possam ser armazenadas, transmitidas e recuperadas pelo destinatário da informação, sem comprometer a sua integridade e confidencialidade. Estas técnicas permitem assim um meio efetivo de proteção de informações suscetíveis a ataques e fornecem formas de comunicação seguras, serviços de autenticação, privacidade e de integridade de dados.

Os serviços básicos de segurança que um sistema criptográfico deve fornecer são, **Confidencialidade**, **Integridade**, **Autenticação** e **Não-Repudiação**. A **Confidencialidade** consiste em manter a informação secreta para todos os que não estão autorizados a ver essa informação. **Integridade** garante que a informação não foi alterada por entidades desconhecidas ou não autorizadas. **Autenticação** garante a identidade de uma entidade envolvida na comunicação. Por último, a **Não-Repudiação** previne a negação de ações e compromissos previamente realizados.

1.1 Enquadramento

Os sistemas/infraestruturas criptográficas mais conhecidas e utilizadas são as de chave pública ou *Public Key Infrastructure* (PKI). Num sistema tradicional, a associação entre um utilizador e a sua chave pública é obtida através de um certificado digital emitido por uma autoridade de certificação ou *Certification Authority* (CA). O CA verifica as credenciais do utilizador antes de emitir o respetivo certificado. Se a Alice quiser enviar uma mensagem cifrada ao Bob, precisa da chave pública do certificado do Bob (Ver **Figura 1.1**):

1. O Bob gera um par de chaves, pública e privada, autentica-se e regista-se no CA;
2. O CA valida a identidade e emite um certificado;
3. A Alice recebe o certificado do Bob:
4. A Alice utiliza a chave pública presente no certificado do Bob para cifrar a mensagem e enviar assim a mensagem cifrada ao Bob;

5. Por sua vez o Bob utiliza a sua chave privada e decifra a mensagem cifrada pela Alice obtendo a mensagem original.

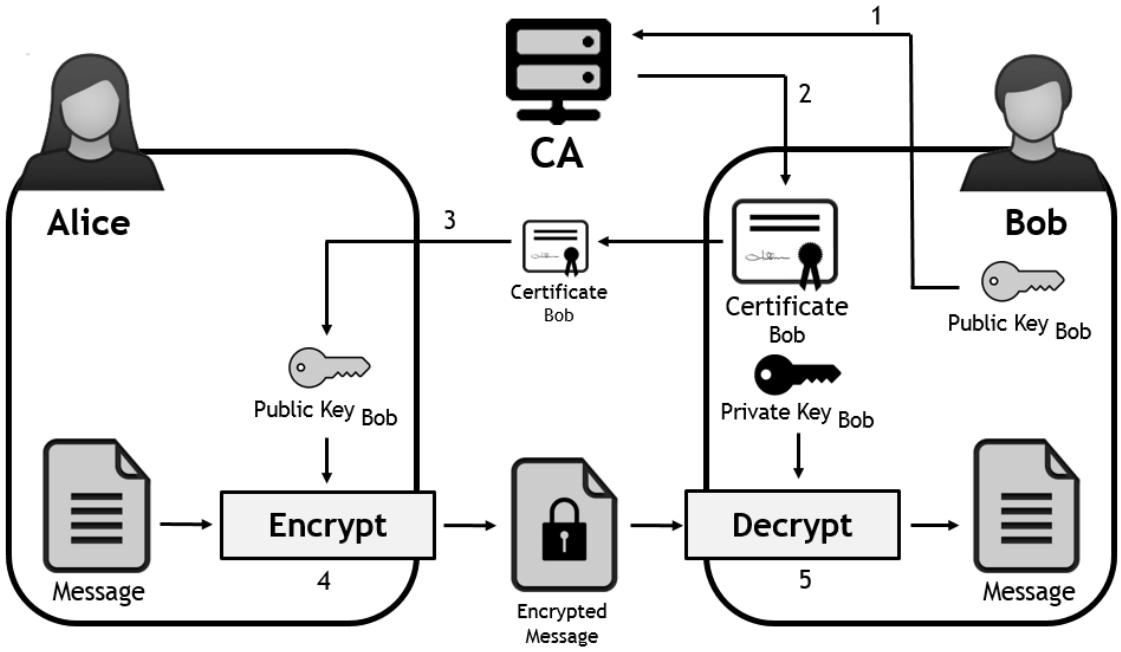


Figura 1.1: Esquema de cifra tradicional com chave pública

De forma simplificar o processo de gestão de certificados, Shamir [Sha85] introduziu o conceito de Criptografia Baseada em Identidade ou *Identity Based Cryptography* (IBC) em 1984. Num sistema de IBC, a chave pública do utilizador é derivada de uma informação de identidade e a sua chave privada é gerada por uma terceira entidade confiável, designada por Gerador de Chave Privada ou *Private Key Generator* (PKG). A principal vantagem deste tipo de sistemas é a simplificação do processo de gestão de chaves. Desta forma, a Alice pode enviar uma mensagem cifrada ao Bob usando a informação de identidade deste, antes de ele obter a sua chave privada do PKG (Ver **Figura 1.2**):

1. A Alice conhece a identidade do Bob, logo irá usar essa identidade como chave pública;
2. A Alice cifra a mensagem utilizando a identidade/chave pública do Bob e envia a mensagem cifrada;
3. O Bob recebe a mensagem e autentica-se ao PKG;
4. O PKG cria a chave privada respetiva e envia ao Bob;
5. O Bob usa a sua chave privada para decifrar a mensagem cifrada pela Alice e obtém a mensagem original.

No caso de assinatura, o Bob pode verificar uma mensagem assinada pela Alice apenas com a informação de identidade dela, sem necessidade de troca de chaves, uma vez que a Alice já conhece a chave privada do Bob (a sua identidade). No geral um sistema de IBC possui as seguintes propriedades:

- a chave pública de um utilizador é a sua identidade (ou derivada da sua identidade);

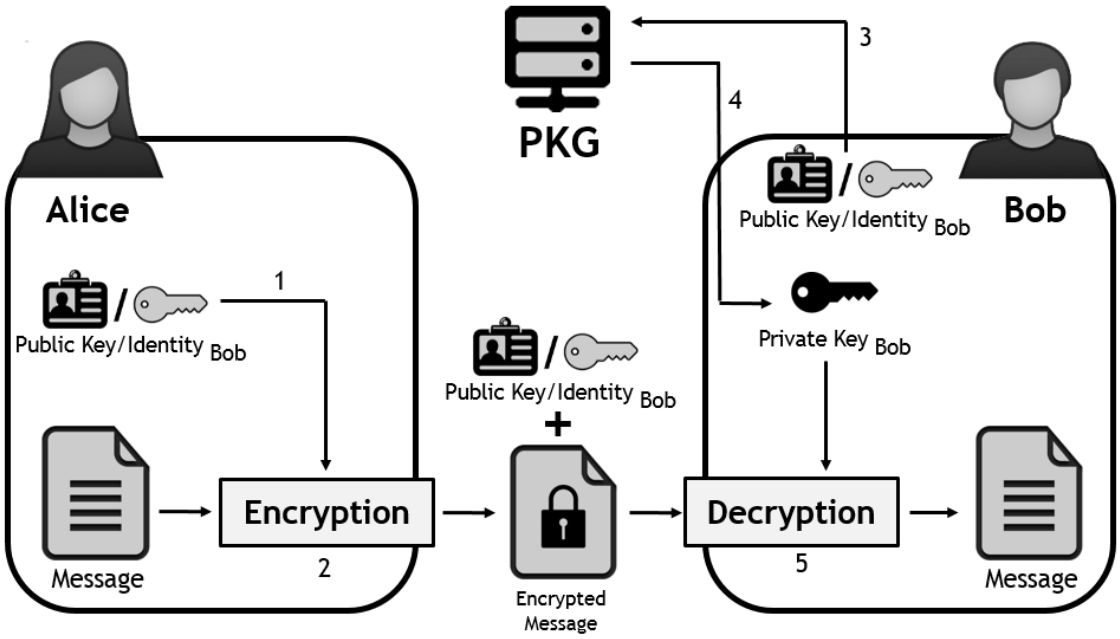


Figura 1.2: Esquema de cifra com Criptografia baseada em Identidade

- não é necessário existir um repositório de chaves públicas;
- a cifra de mensagens e o processo de verificação de assinatura necessitam apenas da identidade do recetor e emissor respetivamente (para além dos parâmetros públicos de sistema[1] gerados pelo PKG).

Estas propriedades fazem dos sistemas IBC vantajosos em relação aos sistemas tradicionais de chave pública, uma vez que a distribuição de chaves é muito simplificada. No entanto, este tipo de sistemas têm a desvantagem do PKG conhecer todas as chaves privadas. Para além disto, é necessário um canal seguro para a emissão de chaves entre o PKG e o utilizador. Um sistema de IBC requer igualmente que o utilizador se autentique ao PKG, da mesma forma que se autenticariam a uma CA.

Como tal, a utilização do cartão de cidadão neste sistema, justifica-se como sendo um elemento crucial para o desenvolvimento. As características de identidade presentes no cartão (necessárias para a chave pública), bem como o seu forte mecanismo de autenticação (necessário para autenticação no PKG, de forma a gerar ou recuperar uma chave privada), associam-se na perfeição às necessidades de um sistema desta natureza.

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*

[1] **parâmetros públicos de sistema**, estes parâmetros são geralmente a chave pública do PKG e os elementos de configuração calculados e publicados pelo PKG (o processo de geração destas elementos só necessita de ser efetuado uma vez).

Environments. Este projeto envolve vários trabalhos para além da presente dissertação e tem como objetivo a criação e disponibilização de soluções de 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 Objetivos

O objetivo principal deste projeto é o desenvolvimento de um sistema criptográfico baseado em características de identidade do utilizador, características essas que estão embutidas no seu cartão eletrónico de identificação (Cartão de Cidadão). É pretendido uma solução que garanta a privacidade e confidencialidade dos dados cifrados, que forneça um nível de confiança forte ao utilizador e que seja flexível e simples, fazendo uso das potencialidades oferecidas pelo Cartão de Cidadão como *Smartcard* e ao mesmo tempo tornando o CC num elemento fulcral para assegurar a segurança do sistema.
Os objetivos principais previstos no plano de trabalho são os seguintes:

- Revisão da literatura sobre os sistemas criptográficos baseados em identidade e estudo dos sistemas de assinatura digital e sistema de cifra;
- Comparação dos sistemas a nível de custo computacional envolvido;
- Definição das características de identidade a utilizar e que estão embutidas no CC;
- Especificação de uma arquitetura baseada num centro de geração de chaves e os respetivos protocolos de distribuição, recuperação e revogação de chaves;
- Implementação de uma aplicação de cifra e decifra de ficheiros, para mostrar a viabilidade da arquitetura proposta;
- Implementação de um sistema simples de políticas de privacidade de ficheiros.

Espera-se que o produto final satisfaça os objetivos e as necessidades requeridas, de forma a comprovar os benefícios de um sistema de criptografia baseada em identidade e que possa mostrar ser uma alternativa viável às arquiteturas tradicionais de chave pública, para a cifra e decifra de mensagens, tanto a nível particular como a nível empresarial.

1.4 Abordagem

Este projeto foi abordado por fases, sendo que o trabalho realizado pode ser dividido em duas fases principais, a fase de Investigação e a fase de Desenvolvimento.

Na fase de Investigação foram estudados os conceitos, componentes e as funcionalidades de *Smartcards*, no caso especifico o CC Português. Foram estudados sistemas criptográficos baseados na identidade, tanto sistemas de assinatura como de cifra e decifra e ainda tecnologias e produtos comerciais já existentes baseados em IBC, de forma a perceber a viabilidade e as potencialidades da arquitetura a desenvolver.

Na fase de Desenvolvimento foram implementados alguns dos esquemas analisados em pequenas aplicações em linguagem *C* para o teste de assinatura e cifra e decifra de pequenas mensagens. Foram criados pequenos *wrappers* para permitir a utilização destes esquemas em linguagens *.NET* e posteriormente criadas algumas aplicações protótipo, bem como a criação de uma biblioteca para facilitar a implementação da arquitetura descrita neste documento.

1.5 Contribuições

No âmbito do projeto desenvolvido, surgiram algumas contribuições que consideramos importantes, nomeadamente:

- Estudo de vários esquemas de IBC, das suas vantagens, potencialidades e problemas em aberto;
- Especificação de uma arquitetura de cifra e decifra de mensagens com criptografia baseada em identidade e respetivos esquemas de geração e recuperação de chaves com o Cartão de Cidadão;
- Aplicação *Desktop* protótipo e respetivo serviço online, (com mecanismos de autenticação através do CC) para geração de chaves;
- Biblioteca para *.NET*, baseada na arquitetura especificada, com todos os métodos necessários para a criação de um sistema completo.

1.6 Estrutura da Dissertação

Este documento está dividido em vários capítulos que mostram sequencialmente o processo de investigação desenvolvido. Neste primeiro capítulo foi descrito o projeto, as motivações, os objetivos que se pretendem alcançar, bem como as contribuições do projeto.

Capítulo 2 - Estado da Arte
Neste capítulo apresenta-se o estudo sobre o Cartão de Cidadão Português e uma introdução à Criptografia Baseada em Identidade, uma apresentação de alguns esquemas importantes, tanto de assinatura como de cifra, bem como os respetivos princípios de segurança, nos quais estes esquemas assentam e aplicações existentes. Este capítulo pretende fornecer uma visão para quem não esteja dentro da problemática tratada neste trabalho.

Capítulo 3 - Algoritmos Implementados
O terceiro capítulo apresenta os algoritmos de IBE implementados, os passos necessários para a implementação dos mesmos no sistema desenvolvido e uma análise de performance computacional e tamanho do texto cifrado dos algoritmos.

Capítulo 4 - Arquitetura
O quarto capítulo aborda a arquitetura tecnológica seguida neste trabalho de uma forma geral. Este capítulo apresenta a estrutura na qual assenta a implementação do sistema criado bem como a justificação das soluções apresentadas, de forma a fornecer uma visão do funcionamento de todo o sistema em cada uma das fases.

Capítulo 5 - Resultados
O quinto capítulo apresenta os resultados obtidos no sistema implementado, as suas funcionalidades e as suas características.

Capítulo 6 - Conclusão e Trabalho Futuro
O último capítulo apresenta as conclusões sobre o trabalho desenvolvido e descreve tecnologias e funcionalidades que podem ser futuramente integradas no sistema, de forma a complementar a solução.

Capítulo 2

Estado da Arte

2.1 Introdução

Neste capítulo é apresentado o estudo do estado da arte e das tecnologias utilizadas no decorrer deste trabalho. Este capítulo começa por apresentar de forma geral o Cartão de Cidadão Português, bem como as suas características e funcionalidades. É ainda apresentada a Criptografia Baseada em Identidade, as suas vantagens, vários tipos de esquemas criptográficos de assinatura e cifra baseados neste tipo de sistemas, bem como os problemas em aberto inerentes.

2.2 Cartão de Cidadão

O Cartão de Cidadão Português começou a ser emitido em Fevereiro de 2007, como documento de cidadania português, com o objetivo de substituir o bilhete de identidade, cartão de contribuinte, cartão de beneficiário 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 do cidadão perante serviços informatizados e a autenticação de documentos eletrónicos.

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 o desenvolvimento das transações eletrónicas através do mecanismo de autenticação forte e da assinatura digital e a interação com diferentes serviços públicos e privados. Assim, permite ao titular provar a sua identidade, perante terceiros, através da autenticação eletrónica. Algumas estatísticas a 13 de agosto de 2013 [1]:

- 8445200 cartões entregues
- 30,47% têm assinatura digital ativada

Visualmente o CC (ver **Figura 2.1**) contém várias informações do cidadão, como a fotografia, elementos de identificação civil e os números de identificação dos diversos organismos públicos, cujo o cartão combina e substitui. Eletronicamente, e como caraterística mais inovadora, é o fato de ser um *smartcard* com diversas funcionalidades [Zuq10], mais especificamente:

- Guardar informação privada, que o titular pode usar mas não conhecer ou divulgar. Esta informação consiste em três criptográficas:
  - Chave simétrica de autenticação do titular;
  - Chave privada de um par de chaves assimétricas RSA para a autenticação do titular;
  - Chave privada de um par de chaves assimétricas RSA para a produção de assinaturas digitais do titular;

[1] Informação retirada do site *cartaodecidadao.pt* a 20 de agosto de 2013

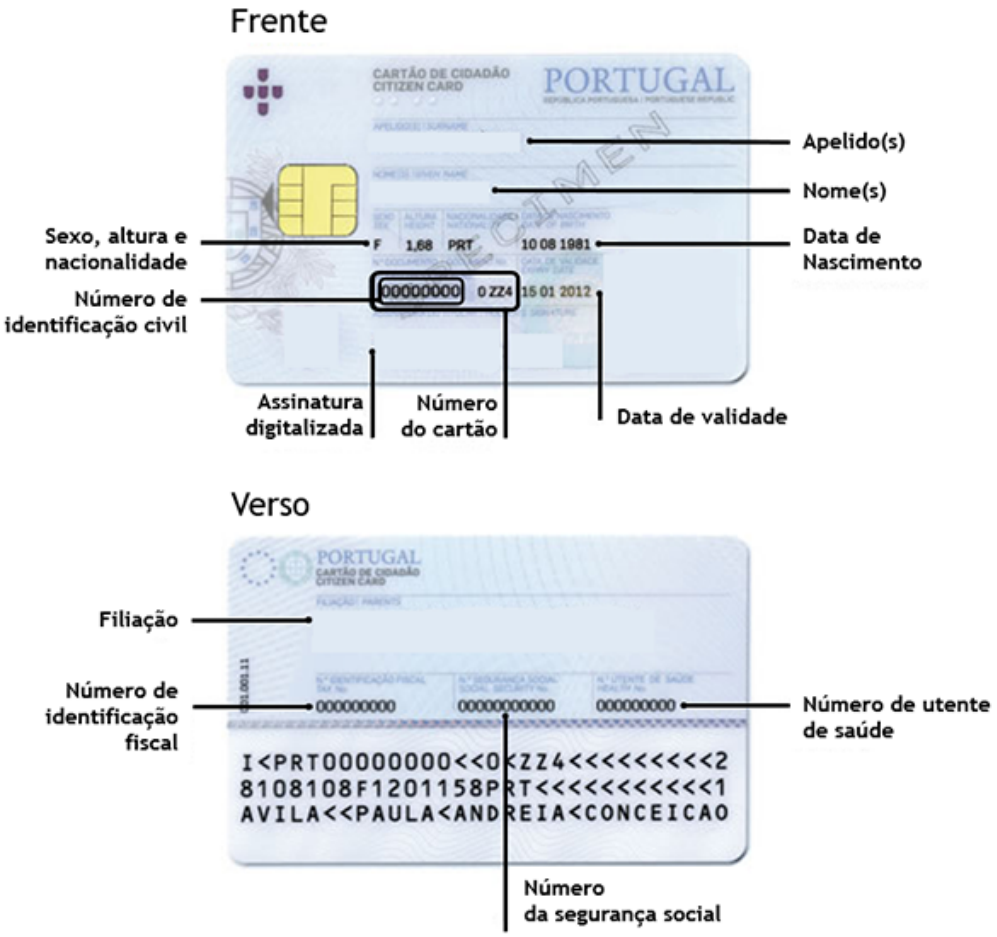


Figura 2.1: Frente e verso do Cartão de Cidadão Português

- Guardar informação pessoal para validação interna de identidade do titular. Esta validação consiste num mecanismo de *Match-on-Card*, que permite comparar uma impressão digital comunicada ao cartão, com a impressão digital do titular (presente no cartão) para fins de validação;
- Guardar informação reservada. Esta informação consiste na morada do titular e pode ser disponibilizada pelo titular de forma fidedigna, através do *smartcard*, a quem desejar ou a quem tiver autorização para obter esta informação;
- Guardar informação pública de grande dimensão. Esta informação consiste na fotografia digital do titular, e pelos certificados de chaves públicas correspondentes, à autenticação e à assinatura digital;
- Guardar toda a informação do titular visível fisicamente no cartão;
- Efetuar operações criptográficas, usando as chaves que fazem parte da informação privada do cartão.

As operações realizadas pelo *smartcard*, em nome do titular do cartão, necessitam da introdução de um *Person Identification Number* (PIN) de quatro algarismos. Cada cartão possui três PIN, para autorização da indicação da morada, autenticação do titular e produção de assinatura digital. Estes PIN são uma camada de segurança, caso o titular perca o seu cartão, pois impede

que as funcionalidades do cartão sejam utilizadas por outrem, uma vez que que cada PIN dispõe apenas de três tentativas de descoberta. Se estas forem utilizadas erradamente o PIN é bloqueado e o seu desbloqueio só é possível junto de uma entidade autorizada e com o *PIN Unlock Key* (PUK) respetivo fornecido com o CC.

2.2.1 Autenticação

A autenticação com *smartcard* do CC pode ser realizada de duas formas. Através do *Europay, Mastercard and Visa Chip Authentication Program* EMV-CAP, em que o cartão é inserido num leitor pessoal e o titular digita o PIN de autenticação de forma a gerar uma One-time Password *OTP*. Esta OTP pode ser enviada a uma entidade que a consiga verificar e assim autenticar o titular.
Outra forma mais comum, é através do par de chaves assimétricas RSA (1024 bits) de autenticação presentes no *smartcard* que podem ser usadas por vários protocolos e aplicações como forma do titular se autenticar. 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 de forma a que possa ser verificado e a chave privada de autenticação validada.

2.2.2 Assinatura

Para a assinatura digital, o *smartcard* do CC possui um par de chaves assimétricas RSA (1024 bits) de assinatura digital, que podem ser usadas para a assinatura de documentos pelo titular. O *smartcard* contém também um certificado com a chave pública da assinatura digital. Este certificado pode ser comunicado a terceiros, com o fim de verificar e validar a assinatura do titular.

Tal como na autenticação, de forma a usar a chave privada do par de chaves assimétricas, o titular do cartão tem que enviar o respetivo PIN de assinatura para o *smartcard*.

2.2.3 Hierarquias e Certificação

Ambos os certificados de chave pública (autenticação e assinatura) dos titulares do CC possuem hierarquias de certificação, que têm como certificado raiz, *GTE CyberTrust Global Root* (ver **Figura 2.2**). Este certificado pertence à empresa *GTE Corporation* e faz normalmente parte dos certificados confiáveis, que vêm instalados por defeito em *browsers* e outro software.

2.2.4 Segurança

A segurança do CC é assegurada por várias características e o *smartcard* obedece a vários padrões e políticas de segurança [MUL11a][MUL11b]. Estes padrões e políticas garantem a salvaguarda da informação privada presente no *smartcard*.

Uma das caraterísticas é a proteção por PIN de controlo de acesso referidos anteriormente na **secção 2.2**. Estes PIN controlam o acesso à morada, chave privada de autenticação e chave privada de assinatura e toleram o máximo de três tentativas. Após três tentativas erradas a funcionalidade respetiva fica inutilizável, e de forma a ser desbloqueada, o titular tem que se deslocar a uma entidade autorizada e indicar o respetivo código de desbloqueio (PUK) que lhe

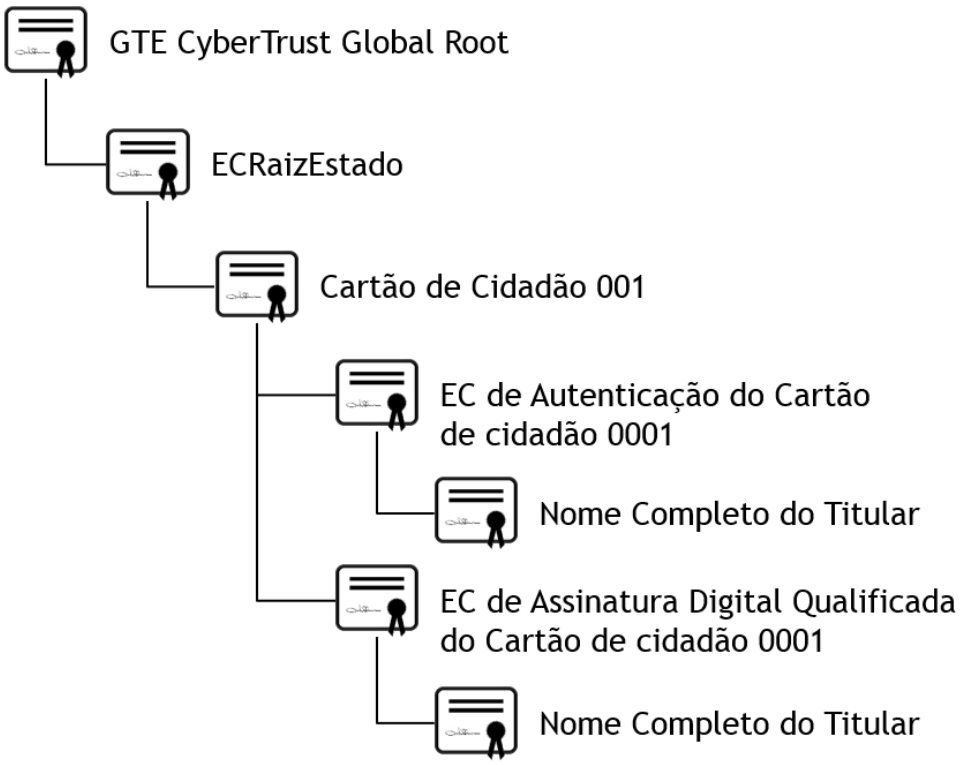


Figura 2.2: Cadeia de certificação dos certificados de chave pública de assinatura e autenticação do Cartão de Cidadão Português

foi fornecido com o CC. Esta funcionalidade de proteção por PIN é especialmente importante para a segurança na utilização do par de chaves assimétricas, pois garante a confidencialidade das chaves privadas, mesmo que o cartão seja extraviado (assumindo que os PIN são apenas do conhecimento do titular do cartão). Para além disto, o CC é fornecido com um código de oito algarismos, que permite o cancelamento de todas as funcionalidades do cartão assim que fornecido às autoridades competentes.
Os certificados de assinatura e de autenticação do titular de um cartão e os respetivos certificados das CA, podem ser validados através do *Online Certificate Status Protocol* (OCSP). Para além disto, todos os certificados, exceto o certificado raiz, incluem informação sobre onde pode ser obtida uma lista de revogação de certificados ou *Certificate Revocation List* (CRL). Esta lista contém todos os certificados que foram revogados e cuja data de validade ainda não expirou, a data de revogação e a razão para a revogação.

2.2.5 Middleware

O *middleware* permite a comunicação entre o sistema operativo do *smartcard* do CC e um computador. O *middleware* para o CC disponibilizado pelo Estado Português [AMA12] é constituído por três *Application Programming Interface* (API):

- CryptoAPI/CSP
- PKCS#11
- eID lib

As duas primeiras interfaces consistem em APIs para operações criptográficas e a *eID lib* para as funcionalidades não criptográficas do CC. As interfaces *CryptoAPI/CSP* e *PKCS#11*, são direcionadas para aplicações baseadas e não baseadas na arquitetura *windows* respetivamente. No caso da interface *eID lib*, esta permite realizar operações com o *smartcard* que são especificas à natureza do CC, como documento de identificação eletrónico. Exemplos destas operações são a obtenção de informação sobre o titular e fotografia, obtenção e alteração da morada, leitura dos certificados e gestão de códigos PIN.

2.3 Criptografia de Curva Elíptica

A utilização de curvas elípticas na criptografia foi sugerida independentemente por Neal Koblitz [Kob87] e Victor Miller [Mil86b] em 1985.
Um curva elíptica é um conjunto de soluções (x, y) para um equação na forma $y^2 = x^3 + Ax + B$, e um ponto extra O, ao qual designamos de ponto no infinito. Para aplicações criptográficas consideramos um campo finito de q elementos: F_q. Onde q é um número primo, assim podemos pensar em F_q como um grupo de inteiros módulo q.
A equação da curva elíptica é normalmente designada por E e é utilizada a notação $E(F_q)$ para um grupo de pontos (x, y) com coordenadas no campo F_q em conjunto com o ponto O (que é definido sobre todos os campos).
Um conjunto de pontos numa curva elíptica forma um grupo sob uma certa regra de adição, à qual damos a notação de $+$. O ponto O é o elemento de identidade do grupo. Quando trabalhamos num campo finito então o grupo é finito (uma vez que há um número finito de pontos).
Dado um ponto $P = (x, y)$ e um inteiro positivo n, definimos $n \cdot P = P + P + ... + P$ (n vezes). A ordem de um ponto $P = (x, y)$ é o menor inteiro positivo n que satisfaça $n \cdot P = O$.

A segurança da criptografia de curva elíptica é baseada no problema de logaritmo discreto de curva elíptica em que:

> E é uma curva elíptica sobre um campo finito F_q;
>
> P é um ponto pertencente a $E(F_q)$ e Q é um ponto pertencente ao grupo $< P >$[2];
>
> **Problema:** encontrar um inteiro t que satisfaça a condição $Q = t \cdot P$

Acredita-se [Mil86b] que o problema do logaritmo discreto de curva elíptica é difícil de resolver computacionalmente quando o ponto P tem como ordem um número primo grande.
A ordem do grupo, normalmente designada como tamanho da curva elíptica, determina a dificuldade do problema. Acredita-se que o mesmo nível de segurança proporcionado por um sistema baseado em RSA com um módulo grande pode ser alcançado por um grupo muito menor de curva elíptica. Assim são reduzidos os requisitos computacionais de armazenamento e transmissão.
Tal como outros sistemas criptográficos populares, não existe até agora uma prova matemática de segurança para criptografia de curva elíptica. Contudo, foi promulgada, bem como sistemas baseados nela, pela *National Security Agency* (NSA)[NSA05] na conferência RSA em 2005 onde recomenda a sua utilização para a proteção de informações tanto classificadas como não-classificadas dos sistemas de segurança e de informação nacionais com chaves de 384 bits. Em

[2] $< P >$ é um grupo gerado por P, ou por outras palavras $< P >= \{O, P, P + P, P + P + P, ...\}$

[BBB+05], o NIST (National Institute of Standards and Technology) apresenta uma tabela em que compara o nível de segurança dos vários algoritmos aprovados tendo como base o tamanho em bits, das chaves (ver **Tabela 2.1**).

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)	Ratio of DH Cost : EC Cost
80	1024	160	3:1
112	2048	224	6:1
128	3072	256	10:1
192	7680	384	32:1
256	15360	512	64:1

Tabela 2.1: Tamanhos de chave recomendados pelo NIST e custos relativos de computação do algoritmo de Diffie-Hellman e Curvas Elípticas

2.4 Criptografia Baseada em Identidade

A Criptografia Baseada em Identidade foi inicialmente proposta em 1984 por Adi Shamir [Sha85], no entanto Shamir apenas conseguiu apresentar uma solução para o esquema de assinatura de mensagens, a cifragem e decifragem de mensagens permaneceu um problema em aberto durante vários anos.

A ideia de Shamir era permitir que uma sequência de bytes correspondente a uma característica que identifique unicamente o utilizador (como por exemplo o nome, número de identificação pessoal ou email) pudesse ser usada como chave pública, criando assim automaticamente um vínculo entre a chave pública e o respetivo dono. Por sua vez, a chave privada seria calculada por uma autoridade na qual os utilizadores teriam que confiar, designada por Gerador de Chave Privada.

IBE Public Key	bob@b.com
RSA Public Key	AAAAB3NzaC1yc2EAAAADAQABAAAAgQDRSFleyASPUMr1RU tzo6++RxQHA228bjcV3vojRQhqR/+1eBAfwqha6fttA5xW PZGevmGeZCNKAR4kriSAHgX5cxj9oMSvB1Ase5SFMptLbm eeOZK4H1JqrO8U07fAIdfj5Y0WZp9pgCnXdmK5/s+dU8C/ dLHTpAfcbx1ioC2sYQ

Tabela 2.2: Comparação entre uma chave pública IBC e uma chave pública RSA

Um sistema criptográfico baseado em identidade, permite que qualquer par de utilizadores possam comunicar entre si de forma segura, sem necessidade de troca prévia de chaves públicas ou de certificados (é eliminado assim o problema de distribuição de chaves públicas) e sem a necessidade de manter repositórios de chaves públicas. Ao contrário dos sistemas tradicionais de criptografia de chave pública, em que esta é geralmente uma cadeia binária de difícil memorização (Ver **tabela 2.2**), em sistemas de IBC o remetente apenas precisa de conhecer a característica identificadora do destinatário da qual irá ser derivada a chave pública, como por exemplo o endereço de email. Esta simplicidade concetual torna este tipo de sistemas espe-

cialmente atraentes, uma vez que faz com que os aspetos criptográficos de uma comunicação sejam praticamente transparentes para o utilizador e possam ser usados pelo utilizador comum, que nada sabe sobre chaves ou protocolos.

No entanto, este tipo de sistemas trazem também algumas desvantagens, as quais serão apresentadas em detalhe nesta secção, bem como as possíveis soluções para ultrapassar estas desvantagens, sem perder as funcionalidades pelas quais os sistemas IBE são conhecidos.

2.4.1 Assinatura Baseada em Identidade

Um esquema genérico de Assinatura Baseada em Identidade ou *Identity Based Signature* (IBS), pode ser descrito em quatro fases/algoritmos aos quais podemos chamar **Setup**, **Extract/KeyGen**, **Sign** e **Verify**, ou **Inicializa**, **Extrai**, **Assina** e **Verifica**. Se a Alice pretender assinar uma mensagem (ver **Figura 2.3**) vai obter do PKG a sua chave privada que está associada à sua chave pública/identidade. A Alice assina a mensagem com a chave obtida. Por sua vez, o Bob utiliza a identidade da Alice para verificar a assinatura da mensagem.

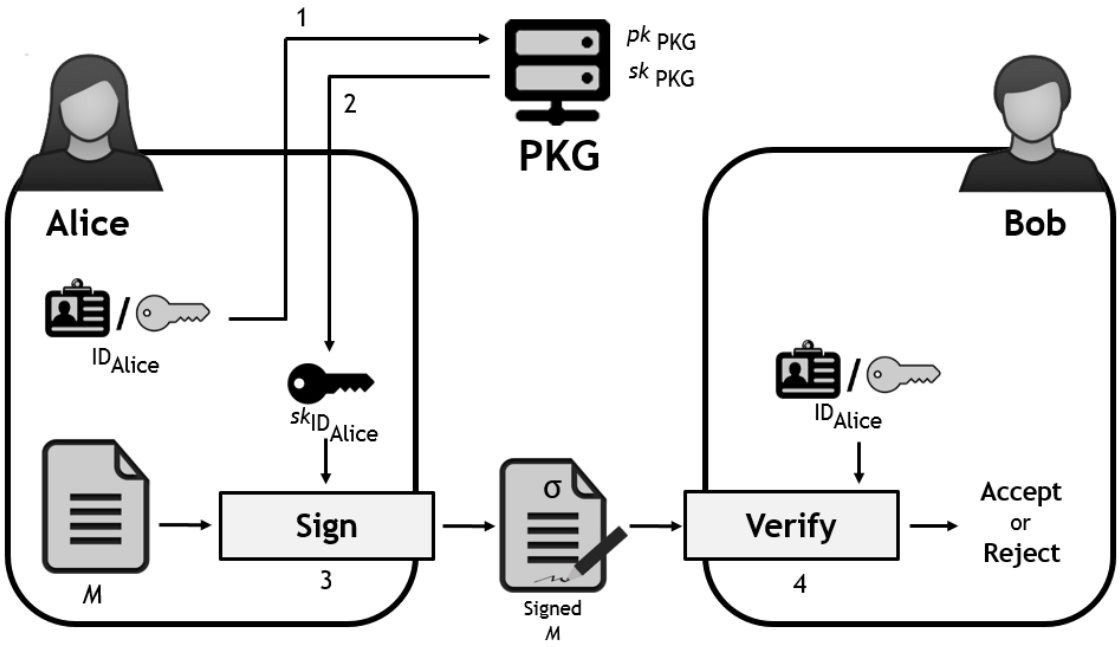


Figura 2.3: Esquema genérico de Assinatura Baseada em Identidade

- **Setup :** O centro de geração de chaves ou PKG cria um par de chaves/parâmetros, chave privada sk_{PKG} e chave pública pk_{PKG} (ambos os intervenientes têm conhecimento desta chave).
- **Extract/Keygen :** Alice autentica-se ao PKG (1) e obtém a sua chave privada (2), $sk_{ID_{Alice}}$ correspondente à sua identidade ID_{Alice}.
- **Sign :** Alice utiliza a sua chave privada $sk_{ID_{Alice}}$, cria a assinatura σ para a mensagem M (3) e envia M assinada.
- **Verify :** Bob ao receber a mensagem M e a assinatura correspondente σ, verifica (4) se a assinatura σ para a mensagem M é genuína, utilizando a identidade da Alice, ID_{Alice} e a chave pública do PKG, pk_{PKG}.

Shamir [Sha85] desenhou e propôs o primeiro esquema de IBS em 1984, baseado no algoritmo RSA (*Rivest, Shamir, Adelman cryptographic algorithm*). No mesmo artigo idealizou o conceito de Cifragem Baseada em Identidade ou *Identity Based Encryption*, no entanto não conseguiu aplicar este conceito na prática.

2.4.2 Cifragem Baseada em Identidade

Em [Sha85] Shamir especifica os requisitos da implementação de um esquema de IBE e lista os dois princípios de implementação:

- A escolha de chaves é baseada num gerador *k* completamente aleatório. Quando o *k* é conhecido, as chaves privadas podem ser facilmente calculadas para uma fração não-negligenciável das chaves públicas possíveis.
- O problema de calcular o gerador *k* a partir de pares de chaves públicas/privadas gerados com *k* é irresolúvel.

Baseado nestes requisitos, Shamir afirma que não é possível criar um esquema de IBE baseado no algoritmo de RSA, mas conjetura que este tipo sistemas criptográficos existem. No entanto o desenvolvimento deste tipo de sistemas foi bastante lento e apesar de terem havido várias propostas para esquemas [Tan88, MY91, TI89, DQ87] ao longo dos anos, nenhuma delas foi considerada como completamente satisfatória.

Em 2000, Joux [Jou04], mostrou que o algoritmo de Victor Miller [Mil86a], para avaliar *Weil pairing* em curvas algébricas, podia ser usado de forma "benéfica"[3] e usou-o num protocolo para construir um acordo de chaves tripartido, baseado no problema de *Diffie-Hellman* em curvas elípticas.
Após o avanço de Joux, em 2001 foi apresentado o primeiro esquema de IBE completamente funcional, eficiente e comprovadamente seguro (dentro do modelo de *random oracles*[4] contra ataques de texto simples escolhido) por Boneh e Franklin [BF01] baseado nas propriedades de *pairing* bilinear em curvas elípticas. Este esquema recebeu especial atenção por ser o primeiro esquema de IBE a ter segurança provável num modelo apropriado de segurança.
Sakai, Ohgishi e Kasahara [RSK01] e Cocks [Coc01] apresentaram igualmente nesse ano, esquemas de IBE. O esquema de Cocks, baseado no problema de resíduos quadráticos, não obteve tanta atenção como o esquema de Boneh e Franklin, pois o método de cifra bit a bit que utiliza, torna o esquema pouco eficiente em termos computacionais e a cifra gera mensagens cifradas bastante longas, quando comparado com outros esquemas (problema de expansão de texto). *Por exemplo, ao cifrar uma mensagem de 128 bits usando um módulo de 1024 bits resulta num texto cifrado com 261424 bits, em comparação, um esquema baseado em pairings, produz um texto cifrado com 288 bits* [BGH07].
Estes problemas foram parcialmente resolvidos por Boneh, Gentry e Hamburg [BGH07], o tamanho do texto cifrado produzido foi reduzido para 1160 bits no caso do exemplo anterior e foi adicionada a propriedade de anonimato ao texto cifrado (ninguém consegue descobrir quem é o recipiente da mensagem ao olhar apenas para o texto cifrado). No entanto, o esquema criado é

[3] O algoritmo de Miller foi inicialmente utilizado para atacar [MVO91, FR94] certos tipos de sistemas criptográficos baseados em curvas elípticas.

[4] Ser seguro no modelo de *random oracles* significa que as funções de *hash* subjacentes usadas no esquema são consideradas funções aleatórias ideais [BR93].

computacionalmente mais exigente que o esquema de Cocks e outros esquemas padrão de IBE e PKI.
Ateniese e Gasti [AG09] basearam-se igualmente no esquema de Cocks, resolveram o problema da eficiência e adicionaram a propriedade de anonimato ao texto cifrado.

Canetti, Halevi e Katz [CHK03] criaram a primeira construção de um esquema IBE com segurança provável, fora do modelo de *random oracles*. Para provar a segurança descreveram um modelo ligeiramente mais fraco de segurança conhecido como modelo de *selective-ID*, no qual o adversário declara que identidade vai atacar, antes dos parâmetros públicos globais serem gerados. Boneh e Boyen [BB04] criaram também um esquema baseado no mesmo modelo, completamente seguro, sem *random oracles*, mas não computacionalmente eficiente para ser usado na prática. Waters [Wat05] apresentou o primeiro esquema eficiente de IBE, completamente seguro, fora do modelo de *random oracles*, em que a segurança é reduzida ao problema de *Decisional Diffie-Hellman Bilinear* (DBDH) (ver **Secção 2.4.2.1**).

A geração de chaves pelo PKG é uma tarefa computacionalmente pesada. Em adição à geração de chaves, o PKG é ainda responsável pela autenticação e pela criação de canais seguros de forma a transmitir as chaves privadas geradas aos respetivos utilizadores. Esquemas de IBE Hierárquicos ou *Hierarchical Identity-Based Encryption* (HIBE) introduzidos por Horwitz e Lynn [HL02], permitem a existência de um PKG raiz que delega a geração de chaves privadas e autenticação de identidade para os PKG de níveis mais baixos. Num HIBE o PKG raiz precisa apenas de ter chaves privadas para os PKG de níveis mais baixos, que por sua vez geram chaves privadas para os utilizadores. Com esta delegação de funções para os PKG de níveis inferiores, a carga computacional da autenticação e geração de chaves é distribuída e reduzida para cada PKG, no entanto, neste esquema é possível que os utilizadores conluiam e obtenham as chave dos PKG de níveis abaixo do primeiro.
Em [GS02], Gentry e Silverberg estendem o esquema de Boneh e Franklin para apresentar uma solução de HIBE completamente resistente a conluio.

Lynn [Lyn02] mostrou a construção de um esquema IBE com autenticação ou *Authenticated Identity-Based Encryption* (AIBE), baseado no esquema de Boneh e Franklin. O recetor verifica a identidade do remetente e se a mensagem foi ou não adulterada, assim é possível remover a necessidade de usar assinaturas digitais, quando a autenticação é necessária.

Sahai e Waters [SW05] desenharam um esquema designado por *Fuzzy* IBE, em que a identidade é vista como um conjunto de atributos descritivos. Este esquema permite que uma chave privada para a identidade ID, possa ser usada para decifrar uma mensagem cifrada com a identidade ID', apenas se ID e ID' estiverem próximas uma da outra, quando medidas por uma distância de semelhança. Esta propriedade de tolerância de erro do esquema, permite o uso de características de identidade biométricas que têm inerente algum ruído, cada vez que são amostradas.

De notar ainda três *Request For Comments* (RFC) nesta área. No *RFC5091* [BM07] os autores descrevem os algoritmos para a implementação dos esquemas Boneh-Franklin e Boneh-Boyen referidos anteriormente. No *RFC5408* [AMS09], Appenzeller *et al*. descrevem a arquitetura de segurança necessária para a implementação de um sistema de IBE. Por último, em [MS09], os autores descrevem convenções para a utilização dos algoritmos descritos no *RFC5091* com *Cryptographic Message Syntax* (CMS).

2.4.2.1 Problemas Base

Com a exceção do esquema de Cocks [Coc01] e outros esquemas derivados deste, baseados no problema de resíduos quadráticos, os esquemas de IBE são na sua maioria baseados em suposições de problemas difíceis em curvas elípticas, as mais utilizadas são:

- **Problema de *Computational Diffie-Hellman* (CDH) em** $\mathbb{G}_1$: não existe um algoritmo eficiente para calcular $\hat{e}(P,P)^{ab}$ a partir $P, aP, bP \in \mathbb{G}_1$ para $a, b \in \mathbb{Z}_q^*$.
- **Problema de *Weak Diffie-Hellman* (W-DH) em** $\mathbb{G}_1$: não existe um algoritmo eficiente para calcular sQ a partir de $P, Q, sP \in \mathbb{G}_1$ e $s \in \mathbb{Z}_q^*$.
- **Problema de *Diffie-Hellman Bilinear* (BDH) em** $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$: não existe um algoritmo eficiente para calcular $\hat{e}(P,P)^{abc} \in \mathbb{G}_2$ a partir de $P, aP, bP, cP \in \mathbb{G}_1$ onde $a, b, c \in \mathbb{Z}_q^*$.
- **Problema de *Decisional Diffie-Hellman Bilinear* (DBDH) em** $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$: não existe um algoritmo eficiente para decidir se $r = \hat{e}(P,P)^{abc}$ dado $r \in \mathbb{G}_2$ e $a, b, c \in \mathbb{Z}_q^*$.

2.4.2.2 Vantagens

A **simplicidade** de um sistema IBC é talvez a sua maior vantagem em relação a outras soluções. Um sistema IBC elimina a necessidade da utilização de certificados, listas de revogação e de estruturas caras e complexas associadas a sistemas PKI. Como resultado, a sua utilização é substancialmente mais simples para o utilizador comum e o seu custo é significativamente mais baixo que o de um sistema PKI. Podemos apontar várias outras potencialidades e vantagens da utilização de sistemas IBC:

Número finito de utilizadores. O sistema pode ser implementado de forma a que, após terem sido distribuídas todas as chaves privadas dos utilizadores, o PKG possa ser fechado e a sua chave privada, utilizada para a geração destas chaves, possa ser destruída.

Não é necessário pré-registo. O utilizador pode receber uma mensagem cifrada, sem nunca ter usado um sistema de cifra. O PKG pode atribuir-lhe uma chave para decifrar a mensagem a pedido. Por outras palavras, um utilizador não precisa de estar registado no sistema para receber mensagens cifradas que a si lhe foram endereçadas.

Expiração de chaves em vez de revogação. Este ponto em esquemas IBE pode ser considerada igualmente como uma vantagem ou desvantagem. Um dos problemas mais difíceis de uma PKI é a revogação de chaves, num esquema IBE podemos utilizar uma data de validade para as chaves em vez de revogação. No entanto esta solução não resolve por completo o problema (ver **Secção 2.4.2.3**).

Vulnerabilidade a *spam* reduzida. Por exemplo, para um sistema PKI para emails é necessária a publicação de uma lista de chaves públicas (repositório de certificados). Uma lista desta natureza pode assim tornar o sistema vulnerável a *spam*. Num esquema de IBE da mesma natureza, estes repositórios não necessitam de ser mantidos, a chave pública de um utilizador é a sua identidade, neste caso o seu email.

Possibilidade de adicionar informação adicional à identidade (chave pública). É possível cifrar uma mensagem para uma certa identidade e especificar por exemplo a data a partir

da qual a chave privada pode ser obtida, a sua validade ou mesmo um intervalo de tempo específico. Outro tipo de informação pode igualmente ser adicionada para forçar, por exemplo, o cumprimento de certas políticas ou características (exemplo: permitir que o utilizador apenas possa recuperar a chave privada respetiva, caso esteja a contactar o PKG dentro de uma gama especifica de IP de uma dada organização).

2.4.2.3 Problemas em Aberto

Revogação de Chaves. Em esquemas tradicionais de PKI, a revogação de chaves públicas é um problema em que os utilizadores para cifrar uma mensagem ou verificar assinatura têm de verificar primeiro se a chave pública foi revogada ou não. Para tal deve ser mantida uma CRL pela PKI. No entanto, com esquemas de IBE ocorre outro tipo de problemas de revogação. Supondo um sistema que utiliza o email como chave pública, em que a identidade do Bob é ***bob@b.com***, se a chave privada correspondente a ***bob@b.com*** for comprometida, o Bob não pode voltar a utilizar o seu mail como chave pública.
Em [BF01] Boneh e Franklin sugerem como solução para este problema adicionar uma *string* temporal à identidade usada como chave pública, em que os utilizadores teriam que renovar as suas chaves privadas periodicamente. Por exemplo, para a identidade do Bob (***bob@b.com***), este usa ***bob@b.com||agosto,2013*** como chave pública. Assim esta chave será apenas válida durante o mês de agosto do ano 2013. No entanto, esta solução não é completa nem prática, uma vez que os formatos do tempo têm que ser definidos à priori e informados aos utilizadores.

Custódia de chaves (*key escrow*). Os esquemas de IBE têm esta propriedade, uma vez que é o PKG que emite as chaves privadas dos utilizadores, usando a chave privada do sistema. Como resultado, o PKG é capaz de assinar ou decifrar qualquer mensagem. Esta propriedade é especialmente indesejável no caso da assinatura, uma vez que a garantia de não-repúdio é um dos requisitos essenciais em esquemas de assinatura digital.
Como solução para este problema, Boneh e Franklin [BF01] sugerem a distribuição da chave privada do PKG (*threshold cryptography*[5]) por vários PKG, utilizando para tal a técnica desenhada por Shamir [Sha79]. Assim, o utilizador obtém partes da sua chave privada de múltiplos PKG e constrói a sua chave privada. Contudo, este sistema tem como desvantagem a carga computacional no utilizador, uma vez que este tem que se autenticar para cada PKG.
No esquema de Baek e Zheng [BZ04], os autores sugerem a partilha de partes da chave privada dos utilizadores, associada à identidade, em vez que partilhar a chave privada do PKG. Este esquema torna-se especialmente atraente em ambientes empresariais. Bob é o presidente do comité, criou a identidade e obteve a chave privada correspondente do PKG. Supondo que o Bob têm que se ausentar, este partilha a sua chave privada por um certo número de servidores, de maneira a que qualquer membro do comité possa decifrar com sucesso a mensagem cifrada, se, e apenas se, obtiver um número mínimo de partes dos servidores. Outra aplicação deste esquema é a construção de um sistema de IBE mediado, em que a ideia é permitir a divisão da chave privada associada a uma identidade em duas partes, dar uma das partes ao utilizador e a outra a um mediador. É possível assim revogar instantaneamente os privilégios do utilizador através de instruções dadas ao mediador. No entanto, o problema de custódia de chaves continua a existir (o PKG conhece a chave privada).
Em [CZKK05], Chen *et al*. apresentam um esquema de IBS baseado em *threshold* e que combina

[5] A ideia é distribuir a informação secreta entre várias entidades de forma a prevenir um único ponto de falha ou de abuso.

as vantagens de um sistema PKI com IBC, que resolve o problema de custódia de chaves e remove as desvantagens do método sugerido por Boneh e Franklin. Shao *et al.* [SCW06] apresentaram mais tarde um esquema mais eficiente e seguro que o apresentado por Chen *et al.*
Em [KG09], Kate e Goldberg apresentam igualmente uma solução baseada em PKG distribuídos que utiliza os esquemas para a geração de chaves de uma forma distribuída, criados por Gennaro *et al.* [GJKR99]. Os autores definem ainda os respetivos protocolos de extração de chaves privadas num modelo de comunicação que pode funcionar de forma síncrona ou assíncrona, para a implementação em três esquemas de IBE, Boneh e Franklin IBE [BF01], Sakai e Kasahara IBE [SK03] e Boneh e Boyen IBE [BB04]. Estes protocolos permitem manter em segredo a chave privada do utilizador dos nodos PKG sem comprometer a sua eficiência.
Os esquemas de HIBE apresentados anteriormente podem também ser utilizados para reduzir o problema de custódia de chaves, dividindo-a por diferentes níveis hierárquicos. Os utilizadores obtém a sua chave privada do PKG correspondente em vez de obterem diretamente a chave do PKG raiz.

2.5 Soluções Existentes

A ideia de Shamir foi inovadora, mas foi o importante avanço de Boneh e Franklin [BF01], apresentado em 2001, que realmente impulsionou a criação de novos esquemas de IBC de cifra e assinatura e a consequente criação de bibliotecas para a implementação destes esquemas e primeiras soluções criptográficas comerciais baseadas nestes.

2.5.1 Soluções Comerciais

Em 2002 o avanço tecnológico de Boneh e Franklin tornou-se na base para a criação da empresa *Voltage Security* por três dos alunos de Boneh: Rauker, Kacker e Appenzeller. Um ano depois em 2003 a *Voltage Security* lança o *Voltage SecureMail* [VSa], uma solução de cifragem de email (com suporte para as principais aplicações e serviços de mail, como por exemplo *Outlook*, *Hotmail*, *Yahoo* e *Gmail*) que se baseia no esquema IBE de Boneh e Franklin para garantir o envio de mensagens de forma segura, sem as dificuldades e as despesas de uma solução tradicional, baseada em certificados. Esta solução baseia-se em usar o email como chave pública (Ver **Figura 2.4**) para cifrar e enviar email de forma segura:

> Passo 1: Alice cifra o email usando o endereço de email do Bob, **bob@b.com**, como chave pública;
>
> Passo 2: Quando o Bob recebe a mensagem, autentica-se ao *Key Server* ou PKG. O PKG contacta um serviço de autenticação externa para autenticar o Bob;
>
> Passo 3: Depois de o Bob estar autenticado junto do PKG, o PKG envia a chave privada do Bob, com a qual ele pode decifrar a mensagem. Esta chave privada pode ser utilizada para decifrar todas as futuras mensagens recebidas pelo Bob.

Algumas estatísticas [VSb] do *Voltage SecureMail* em 2011 incluíam 50 milhões de utilizadores mundiais (com projeção de 100 milhões em 2014), estimativa de aproximadamente mil milhões de emails cifrados e enviados em 2011. Um terço das 20 maiores companhias públicas mundiais usavam esta solução como standard para o envio de emails seguros.

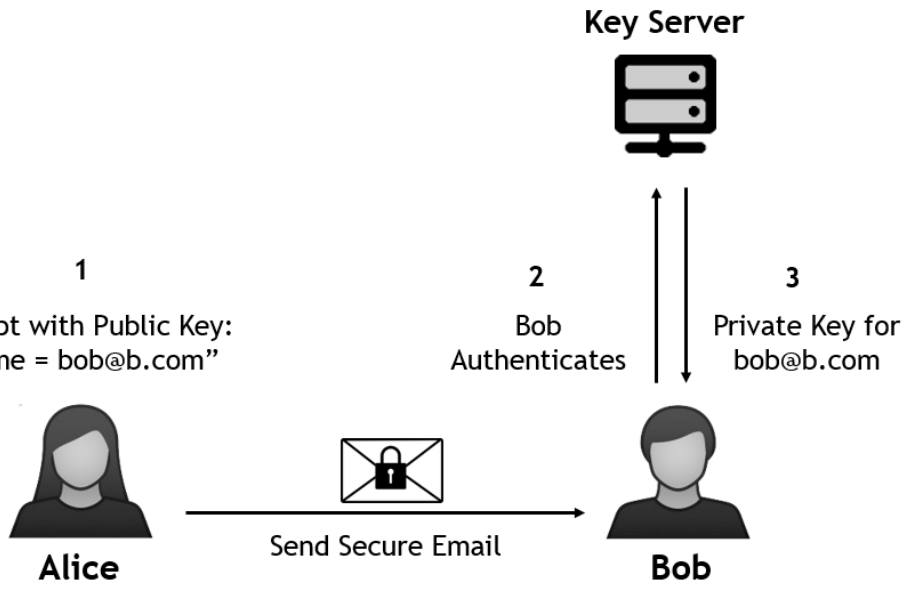


Figura 2.4: Esquema de IBE para email da *Voltage Security*

Investigadores da *Hewlett Packard* (HP) [MBDH] desenvolveram um sistema de informação para confidencialidade e privacidade de mensagens baseado em IBE (esquema de Boneh e Franklin) num contexto de serviços de saúde, como hospitais e outras organizações. Este sistema foi aplicado para avaliação num contexto real, com o objetivo de fornecer uma alternativa segura e simples aos esquemas tradicionais de PKI.
Esta solução foi aplicada numa infraestrutura já existente, com requisitos predefinidos, nomeadamente a utilização de clientes e servidores de mail e mecanismos de autenticação pré-existentes. O sistema em si consiste na modificação da infraestrutura de email para permitir o envio de mensagens confidenciais entre utilizadores da *intranet* do serviço de saúde. Assim, permite o envio de mensagens confidenciais para endereços e listas de email acessíveis por diferentes tipos de utilizadores, com diferentes papéis e direitos dentro da organização, onde apenas utilizadores com o papel necessário (selecionado pelo remetente) dentro da organização têm acesso ao conteúdo do email.

Outras soluções de email existentes baseadas em IBE incluem a solução *FortiMail* [For] da empresa *Fortinet* e a *Trend Micro Email Encryption* [TM] da empresa *Trend Micro* baseado no esquema de Sakai e Kasahara [SK03]. De notar igualmente a solução desenvolvida pela empresa *Gemalto* [Gem] que consiste no primeiro sistema de IBE para *smartcards*, baseado no esquema de Boneh e Franklin.

2.5.2 Bibliotecas

Existem várias bibliotecas *open source* disponíveis que auxiliam na implementação de esquemas de assinatura e cifra baseados em IBC em várias linguagens de programação, nomeadamente bibliotecas para a criação e utilização de curvas elípticas:

- **PBC Library** [Lyn08] - biblioteca para a linguagem *C* desenvolvida por Ben Lynn, um ex-aluno de Boneh, com várias soluções para esquemas IBE publicadas. Contém várias implementações e exemplos de esquemas de IBS e IBE, e vários programas para geração de

curvas elípticas.

- **jPBC** [DCI11] - biblioteca para a linguagem *Java*, consiste num *port* e num *wrapper* da biblioteca PBC Library. Corre na plataforma *Android*, sem ser necessária qualquer alteração à biblioteca. Tal como a PBC Library, inclui vários exemplos de esquemas.
- **MIRACL** [Cer] - biblioteca para as linguagens *C* e *C++*, criada pela empresa *Shamus Software* e adquirida em 2011 pela empresa *CertiVox*. Tem como principal característica o fato de correr em vários ambientes, incluindo plataformas móveis.
- **Charm Crypto** [AGM+13] - biblioteca para a linguagem *Python* que tem por base módulos em linguagem *C*. Foi desenhada para minimizar o tempo de desenvolvimento e a complexidade do código.

2.6 Conclusões

Foram referidas várias tecnologias ao longo deste capítulo. O Cartão de Cidadão e as suas funcionalidades como documento de identificação eletrónico e *smartcard*, permitem ao cidadão identificar-se de forma segura perante os serviços informatizados, através dos mecanismos de autenticação e a assinatura de mensagens e documentos eletrónicos.

Foi abordada igualmente a criptografia sobre curva elíptica e a sua aplicação em sistema criptográficos baseados em identidades. Um sistema criptográfico baseado em identidade, permite que qualquer par de utilizadores possa comunicar entre si de forma segura, sem necessidade de troca prévia de chaves públicas ou de certificados e sem a necessidade de manter repositórios de chaves públicas. A simplicidade deste tipo de sistemas, elimina a necessidade da utilização de certificados, listas de revogação e de estruturas caras e complexas, associadas a sistemas PKI. Foram apresentados igualmente os problemas em aberto, inerentes deste tipo de sistemas, como a revogação e a custódia de chaves.

O objetivo deste capítulo prende-se à utilização e combinação das soluções e tecnologias apresentadas na criação de um protótipo de uma arquitetura criptográfica para ficheiros, que pretende ser inovadora, segura, viável, simples de implementar e fácil na sua utilização, para o utilizador comum. Esta arquitetura não pretende ser um concorrente às várias soluções existentes para infraestruturas criptográficas, desenvolvidas por grandes organizações, mas sim apresentar-se como um sistema sólido e seguro, capaz de demonstrar a viabilidade da nossa solução para utilização por pequenas organizações.

Capítulo 3

Implementação dos Algoritmos

3.1 Introdução

Neste capítulo é feita uma descrição dos dois esquemas de IBE implementados, são apresentadas as fases da implementação destes esquemas em linguagem *C*, bem como os passos necessários para a utilização dos mesmos na linguagem *.NET C#*. É feita uma comparação dos dois esquemas em termos de análise de performance computacional e tamanho do texto cifrado dos algoritmos. É também apresentada uma biblioteca baseada no esquema selecionado na análise para *.NET C#* para a criação de infraestruturas de cifra e decifra de ficheiros e recuperação e geração de chaves privadas.

3.2 Esquemas de IBE e IBS

De forma a criar uma arquitetura de cifra e decifra de ficheiros baseada em IBE, e protótipos iniciais, foi necessário numa fase inicial a seleção de um esquema de IBE seguro, eficiente e adequado às necessidades da arquitetura. Foram selecionados dois esquemas, o esquema *Fullident* (*BF Fullident*) proposto por Boneh e Franklin [BF01] em 2001, e *Multi-Receiver IBE* (*Emribe*), proposto por Baek *et al*. [BSNS05] em 2005. A escolha do esquema *BF Fullident*, é justificada pela importância do mesmo para a literatura de IBE. No caso do esquema *Emribe*, a escolha incidiu sobre este, pois implementa a cifra de uma mensagem para *n* recetores/utilizadores por defeito e de forma eficiente.
Foi igualmente selecionado um esquema de assinatura IBS. Esta escolha incidiu sobre o esquema criado por Paterson [Pat02] em 2002 (*Paterson IBS*) pelo fato de utilizar as mesmas primitivas computacionais subjacentes aos esquemas *BF Fullident* e *Emribe*, isto possibilita que possam ser usados os mesmo parâmetros de sistema e as mesmas chaves.

3.2.1 Esquema *BF Fullident*

Setup: Dado um número primo q, o algoritmo funciona da seguinte forma:

Passo 1: Selecionar dois grupos $\mathbb{G}_1$, $\mathbb{G}_2$ de ordem q, e um mapa bilinear que satisfaça $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Escolher um gerador aleatório $P \in \mathbb{G}_1$.

Passo 2: Escolher um número aleatório $s \in \mathbb{Z}_q^*$ e calcular $P_{pub} = s \cdot P$.

Passo 3: Escolher uma função de *hash* criptográfico $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$. Escolher uma função de *hash* criptográfico $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$ para algum n. O espaço da mensagem cifrada $\mathcal{C} = \mathbb{G}_1^* \times \{0,1\}^n$. Os parâmetros do sistema são params = $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, H_1, H_2 \rangle$. A master-key é $s \in \mathbb{Z}_q^*$.

Passo 4: Escolher uma função de *hash* criptográfico $H_3 : \{0,1\}^n \times \{0,1\}^n \rightarrow \mathbb{Z}_q^*$ e uma função de *hash* criptográfico $H_4 : \{0,1\}^n \rightarrow \{0,1\}^n$.

Símbolo	Significado
$\mathbb{Z}$	grupo de inteiros
$\mathbb{Z}_q$	grupo {0,...,q-1} sob adição de módulo q
$\mathbb{Z}_q^+$	grupo multiplicativo de inteiros módulo número primo q
$\mathbb{G}$	grupo de ordem primo
$\mathcal{G}$	algoritmo gerador que satisfaz a hipótese de *Diffie-Hellman (DH) Bilinear*
d_{ID}	chave privada de ID
F_{ID}	chave pública de ID
s	chave privada / chave mestra do sistema
P, P_{pub}	parâmetro público de sistema
H_i	função de *hash*

Tabela 3.1: Notações do Esquema *BF Fullident*

Extract: Para uma dada string ID $\in \{0,1\}^*$ o algoritmo: (1) calcula $F_{\text{ID}} = H_1(\text{ID}) \in \mathbb{G}_1^*$, e (2) define a chave privada como $d_{\text{ID}} = s \cdot F_{\text{ID}}$ onde s é a chave privada/chave mestra do sistema.

Encrypt: Para cifrar a mensagem $M \in \{0,1\}^n$ com a chave pública ID: (1) calcular $F_{\text{ID}} \in \mathbb{G}_1^*$, (2) escolher um número aleatório $\sigma \in \{0,1\}^n$, (3) calcular $r = H_3(\sigma, M)$ e definir a mensagem cifrada como:

$$C = \langle r \cdot P,\ \sigma \oplus H_2(g_{\text{ID}}^r),\ M \oplus H_4(\sigma) \rangle \text{ onde } g_{\text{ID}} = \hat{e}(F_{\text{ID}}, P_{\text{pub}}) \in \mathbb{G}_2^*$$

Decrypt: Seja $C = \langle U, V, W \rangle \in \mathcal{C}$ uma mensagem cifrada usando a chave pública ID, se $U \notin \mathbb{G}_1^*$ rejeitar a mensagem cifrada. Para decifrar C usando a chave privada $d_{\text{ID}} \in \mathbb{G}_1^*$:

1. Calcular $V \oplus H_2(\hat{e}(d_{\text{ID}}, U)) = \sigma$.
2. Calcular $W \oplus H_4(\sigma) = M$.
3. Definir $r = H_3(\sigma, M)$. Verificar se $U = r \cdot P$. Caso não se verifique, rejeitar a mensagem cifrada.
4. M é a mensagem decifrada resultante de C.

A cifra de M, $W = M \oplus H_4(\sigma)$ pode ser substituída por $W = E_{H_4(\sigma)}(M)$ onde E é um esquema de cifra simétrica.

3.2.2 Esquema *Emribe*

Setup: Dado um número primo q, o algoritmo funciona da seguinte forma:

Passo 1: Selecionar dois grupos $\mathbb{G}_1$ e $\mathbb{G}_2$ de ordem q, e um mapa bilinear que satisfaça $\hat{e}$: $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.

Passo 2: Escolher dois números aleatórios $Q \in \mathbb{G}_1$ e $P \in \mathbb{G}_1$.

Passo 3: Escolher um número aleatório $s \in \mathbb{Z}_q^*$ e calcular $P_{\text{pub}} = s \cdot P$. A master-key é s.

Passo 4: Selecionar as funções de *hash* criptográfico $H_1 : \{0,1\}^* \to \mathbb{G}_1$, $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ para algum n, e $H_3 : \mathbb{G}_1 \times \cdots \times \mathbb{G}_1 \times \mathbb{G}_2 \times \{0,1\}^n \to \{0,1\}^n$.

Símbolo	Significado
$\mathbb{Z}$	grupo de inteiros
$\mathbb{Z}_q$	grupo {0,...,*q*-1} sob adição de módulo *q*
$\mathbb{Z}_q^+$	grupo multiplicativo de inteiros módulo número primo *q*
$\mathbb{G}$	grupo de ordem primo
$\mathcal{G}$	algoritmo gerador que satisfaz a hipótese de *DH Bilinear*
d_{ID_i}	chave privada de ID_i
F_{ID_i}	chave pública do ID_i
s	chave privada / chave mestra do sistema
P, P_{pub}, *Q*	parâmetro público de sistema
H_i	função de *hash*

Tabela 3.2: Notações do Esquema *Multi-Receiver IBE*

Extract: Para uma dada string $ID \in \{0,1\}^*$, calcular $d_{ID} = s \cdot F_{ID} \in \mathbb{G}_1^*$, em que F_{ID} é a chave pública, calculada como $F_{ID} = H_1(ID) \in \mathbb{G}_1^*$. Devolver d_{ID} como chave privada associada à identidade ID.

Encrypt: Para cifrar a mensagem $M \in \{0,1\}^n$ com múltiplas identidades/chaves públicas ($ID_1,...,ID_n$), escolher dois números aleatórios, $R \in \mathbb{G}_2$ e $r \in \mathbb{Z}_q^*$, e calcular $C = (U, V_1,...,V_n,W1,W2,\mathcal{L},\sigma)$ de forma a que

$$(U, V_1, \ldots, V_n, W1, W2, \mathcal{L}, \sigma) = (r \cdot P, r \cdot F_{ID_1} + r \cdot Q, \ldots, r \cdot F_{ID_n} + r \cdot Q, \hat{e}(Q, P_{pub})^r R, M \oplus H_2(\cdot R), H_3(R, M, U, V_1, \ldots, V_n, W1, W2, \mathcal{L}))$$

Devolve *C* como mensagem cifrada, σ garante a integridade da sequência inteira da mensagem cifrada e $\mathcal{L}$ contém informação sobre as identidades para o qual foi cifrada. $W2 = M \oplus H_2(R)$ pode ser substituída por $W2 = E_{H_2(R)}(M)$ onde *E* é um esquema de cifra simétrica.

Decrypt: Para decifrar a mensagem para uma dada identidade ID_i em que $i \in [1,n]$: analisar *C* como $(U, F_{ID_1}, \ldots, F_{ID_n}, W1, W2, \mathcal{L}, \sigma)$, e utilizando a informação presente em $\mathcal{L}$ encontrar o F_{ID_i} apropriado. Calcular $R = \frac{\hat{e}(U, d_{ID_i})}{\hat{e}(P_{pub}, F_{ID_i})} \cdot W1$, $M = W2 \oplus H_2(R)$, e $\sigma' = H_3(R, M, U, V_1, \ldots, V_n, W1, W2, \mathcal{L})$). Se $\sigma' = \sigma$ aceitar *M* como mensagem decifrada, caso contrário, rejeitar.

3.2.3 Esquema de Assinatura *Paterson IBS*

Setup: Dado um número primo *q*, o algoritmo funciona da seguinte forma:

Passo 1: Selecionar dois grupos $\mathbb{G}_1 = \langle P \rangle$ e $\mathbb{G}_2$ de ordem *q*, e um mapa bilinear que satisfaça $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.

Passo 2: Escolher um número aleatório $P \in \mathbb{G}_1$.

Passo 3: Escolher um número aleatório $s \in \mathbb{Z}_q^*$ e calcular $P_{pub} = s \cdot P$. A master-key é *s*.

Passo 4: Selecionar as funções de *hash* criptográfico $H_1 : \{0,1\}^* \to \mathbb{G}_1$, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$, e $H_3 : \mathbb{G}_1 \to \mathbb{Z}_q$.

Símbolo	Significado
$\mathbb{Z}$	grupo de inteiros
$\mathbb{Z}_q$	grupo {0,...,*q*-1} sob adição de módulo *q*
$\mathbb{Z}_q^+$	grupo multiplicativo de inteiros módulo número primo *q*
$\mathbb{G}$	grupo de ordem primo
$\mathcal{G}$	algoritmo gerador que satisfaz a hipótese de *DH Bilinear*
d_{ID_i}	chave privada de ID_i
F_{ID_i}	chave pública do ID_i
s	chave privada / chave mestra do sistema
P, P_{pub}	parâmetro público de sistema
H_i	função de *hash*

Tabela 3.3: Notações do Esquema de Assinatura *Paterson IBS*

Extract: Para uma dada string $ID \in \{0,1\}^*$, calcular $d_{ID} = s \cdot H_1(ID) \in \mathbb{G}_1^*$. Devolver d_{ID} como chave privada associada à identidade ID.

Sign: Para assinar a mensagem $M \in \{0,1\}^n$, escolher o número aleatório $r \in \mathbb{Z}_q^*$ (r^{-1} é o inverso de *r* em $\mathbb{Z}_q^*$), e calcular a assinatura de *M* como o par $(R,S) \in \mathbb{G}_1 \times \mathbb{G}_1$, onde:

$$R = r \cdot P, \;\; S = r^{-1} \cdot (H_2(M) \cdot P + H_3(R) \cdot d_{ID})$$

Verify: Para verificar a assinatura (*R*,*S*) na mensagem *M*, calculamos $\hat{e}(R,S)$ e comparamos com $\hat{e}(P,P)^{H_2(M)} \cdot \hat{e}(P_{pub}, F_{ID})^{H_3(R)}$ (onde F_{ID} é $H_1(ID) \in \mathbb{G}_1^*$). A assinatura é aceite se estes dois valores coincidirem em $\mathbb{G}_2$, pois se (*R*,*S*) for uma assinatura válida em *M*, temos:

$$\begin{aligned}\hat{e}(R,S) &= \hat{e}(r \cdot P,\, r^{-1} \cdot (H_2(M) \cdot P + H_3(R) \cdot d_{ID})) \\ &= \hat{e}(P,\, H_2(M) \cdot P + H_3(R) \cdot d_{ID}) \\ &= \hat{e}(P,\, P)^{H_2(M)} \cdot \hat{e}(P_{pub},\, F_{ID})^{H_3(R)}\end{aligned}$$

3.3 Implementação

De forma a implementar os esquemas apresentados, houve necessidade de recorrer a uma biblioteca que permitisse tanto a geração de curvas elípticas como as operações sobre estas, como por exemplo *pairings*. Para tal utilizámos a biblioteca para linguagem *C* apresentada na **Secção 2.5.2**), *PBC Library* [Lyn08].

3.3.1 Implementação em *C*

A implementação dos esquemas de IBE foi feita numa primeira fase sob a forma de aplicações teste em linguagem *C*, exclusivamente para a cifra e decifra de pequenas mensagens. Numa segunda fase foram criados *wrappers* para a implementação dos esquemas (IBE e IBS) na linguagem *.NET C#* e posteriormente uma biblioteca nesta linguagem com o esquema de IBE escolhido, para utilização no desenvolvimento da arquitetura de cifra e decifra de ficheiros baseada em IBC, com o CC como mecanismo de autenticação do utilizador no PKG.

3.3.1.1 Aplicações de Teste

Foram implementados ambos os esquemas de IBE em aplicações simples de funcionalidade limitada, para demonstrar a cifra de uma mensagem curta com uma identidade *X* e a respetiva decifra da mensagem, utilizando a chave privada correspondente a essa identidade. Estas aplicações geram novos parâmetros de sistema sempre que são executadas, servindo unicamente como teste aos *wrappers* e ao respetivo esquema.

3.3.1.2 *Wrapper* DLL

A ideia inicial do projeto foi adicionar capacidades criptográficas ao CC usando métodos de IBC. Decidiu-se usar *.NET C#* como linguagem de desenvolvimento e usar assim as bibliotecas .NET de desenvolvimento para o CC.
Uma vez que, tanto quanto é do nosso conhecimento, não existem bibliotecas para *C#* que permitam a utilização de curvas elípticas e efetuação de *pairings* entre elementos, foi necessário criar um *wrapper* sob a forma de uma *Dynamic-Link Library* (DLL) que nos permitisse usar as funções da biblioteca PBC Library em *C#*.

pkg.dll	
Setup	
Generate s (master key)	s = *generateNewMasterKey(curve, (...))*
Generate P parameter	P = *generateNewPparam(curve, (...))*
Calculate $P_{pub} = s \cdot P$	P_{pub} = *getPpub(curve, P, s, (...))*
Extract	
Generate $d_{ID} = s \cdot F_{ID}$	d_{ID} = *getPrivateKey(curve,* F_{ID}*, s, (...))*
client.dll	
Encrypt	
Get element r	r = *encryptGetR(curve,* $H_3(\sigma,M)$*, (...))*
Calculate $U = r \cdot P$	U = *encryptCalcU(curve, P, r, (...))*
Calculate $H_2(g_{ID}^r)$	H_2 = *encryptCalcH2(curve, r,* F_{ID}*,* P_{pub}*, (...))*
Decrypt	
Calculate $H_2(\hat{e}(d_{ID},U))$	r = *decryptCalcH2(curve,* d_{ID}*, U, (...))*

Tabela 3.4: *Wrappers* para PKG e cliente do esquema *BF Fullindent* (ver **Anexo A.1**)

Foram criados dois *wrappers* para os dois esquemas de IBE apresentados, BF Fullident e Emribe (ver **Anexos A.1** e **A.2**). Estes *wrappers* estão divididos em dois módulos correspondentes ao utilizador e ao PKG e implementam exclusivamente as operações sobre curva elíptica no esquema (as restantes operações podem ser efetuadas normalmente em linguagem *C#*) necessárias pelo utilizador e pelo PKG respetivamente (o utilizador não necessita das funções do PKG nem vice-versa[1]).

[1] A segurança do esquema não é dependente do acesso do utilizador às funções presentes no *wrapper* do PKG, a separação em dois *wrappers* tem como objetivo simplificar a implementação de um sistema IBE.

Por si só os *wrappers* não implementam a cifra e decifra de mensagens, mas permitem a utilização das funções necessárias para a criação de um sistema com essas capacidades em *C#*. Assim temos dois DLLs, um para implementação numa aplicação cliente e outro para utilização no PKG, cada um com as funções especificas necessárias para efetuar operações sobre curvas elípticas.
Nas **Tabelas 3.4** e **3.5** é possível observar a correspondência para o esquema *BF Fullident* e *Emribe*, respetivamente entre os cálculos necessários sobre a curva elíptica nas quatro fases (ver **Secção 3.2.1**) e as operações correspondentes no *wrapper*. Assim, as funções no *wrapper* do PKG têm como fim gerar os parâmetros públicos e privados do sistema (**Setup**) e gerar a chave privada (**Extract**). Por sua vez, o *wrapper* do cliente/utilizador tem por função gerar os elementos necessários para a cifra e decifra (**Encrypt** e **Decrypt**).

pkg.dll	
Setup	
Generate s (master key)	*s = generateNewMasterKey(curve, (...))*
Generate P parameter	*P = generateNewPparam(curve, (...))*
Generate Q parameter	*Q = generateNewQparam(curve, (...))*
Calculate $P_{pub} = s \cdot P$	P_{pub} *= getPpub(curve, P, s, (...))*
Calculate $\hat{e}(Q,P_{pub})$	$\hat{e}(Q,P_{pub})$ *= calcpairQPpub(curve, Q,* P_{pub}*, (...))*
Extract	
Generate $d_{ID} = s \cdot F_{ID_i}$	d_{ID} *= getPrivateKey(curve,* F_{ID_i}*, s, (...))*

client.dll	
Encrypt	
Generate random element r	*r = genSmallR(curve, (...))*
Generate random element R	*R = genBigR(curve, (...))*
Calculate $U = r \cdot P$	*U = calcU(curve, P, r, (...))*
Calculate $rQ = r \cdot Q$	*rQ = calcrQ(curve, Q, r, (...))*
Calculate $V = r \cdot F_{ID_i} + r \cdot Q$	*V = calcV(curve, r,* F_{ID_i}*, rQ, (...))*
Calculate $W_1 = \hat{e}(Q,P_{pub})^r \cdot R$	W_1 *= calcW1(curve,* $\hat{e}(Q,P_{pub})$*, r, R, (...))*
Decrypt	
Calculate $R = \frac{\hat{e}(U,d_{ID_i})}{\hat{e}(P_{pub},F_{ID_i})} W_1$	*R = decryptCalcR(curve, U,* d_{ID}*,* P_{pub}*,* F_{ID_i}*,* W_1*, (...))*

Tabela 3.5: *Wrappers* para PKG e cliente do esquema *Emribe* (ver **Anexo A.2**)

A utilização destes *wrappers* num sistema real, começa pela escolha ou geração de uma curva elíptica por parte do PKG. Depois da escolha da curva pode ser gerada pelo PKG a chave privada (***s***) para ambos os esquemas, os parâmetros públicos, $\boldsymbol{P}$ e $\boldsymbol{P}_{pub}$ para o esquema *BF Fullident* e $\boldsymbol{P}$**,** $\boldsymbol{Q}$**,** $\boldsymbol{P}_{pub}$ e $\boldsymbol{\hat{e}(Q,P_{pub})}$ (este cálculo prévio evita que o utilizador efetue uma operação extra durante a cifra) para o esquema *Emribe*. A definição da curva elíptica utilizada pelo PKG, bem como os parâmetros públicos, têm que ser disponibilizados ao utilizador/aplicação cliente, enquanto que a chave privada será do conhecimento exclusivo do PKG, nela dependerá a segurança do sistema.

Para além dos *wrappers* descritos, foi ainda criado outro, para a implementação do esquema

de IBS, *Paterson IBS* (ver **Anexo A.3**). Este *wrapper* consiste num único módulo e é utilizado exclusivamente pelo utilizador. Como os parâmetros e as chaves necessárias para assinatura e verificação neste esquema utilizam as mesmas primitivas computacionais que o esquemas *BF Fullident* e *Emribe*, podemos utilizar os parâmetros de sistema e as chaves privadas geradas pelo PKG de um destes esquemas. Este aspeto torna-se bastante conveniente, pois permite adicionar propriedades de assinatura e verificação de mensagens no lado do utilizador/cliente, a um sistema que já tenha implementado um destes esquemas IBE, de forma simples e eficiente sem necessidade de efetuar qualquer alteração no PKG existente.

sign.dll	
Setup	
Generate *r*	r = *generateSignr(curve, (...))*
Calculate *R* element	R = *calcSignR(curve, r, P, (...))*
Calculate *S* element	S = *calcSignS(curve, r, P,* $H_2(M)$, d_{ID}, *R, (...))*
Verify	
Verify Signature	*ver = verifySign(curve, P,* P_{pub}, $H_2(M)$, F_{ID}, *R, S, (...))*

Tabela 3.6: *Wrapper* para assinatura, esquema *Paterson IBS* (ver **Anexo A.3**)

Na **Tabela 3.6** é possível observar a correspondência para o esquema *Paterson IBS*, entre os cálculos necessários sobre a curva elíptica para a criação e verificação da assinatura e as operações correspondentes no *wrapper*.

3.3.2 Implementação em *.NET C#*

Partindo dos *wrappers* descritos anteriormente, desenvolvemos aplicações para os dois esquemas em *.NET C#* com capacidades para cifrar e decifrar ficheiros. Para permitir estas capacidades, e uma vez que não é viável cifrar um ficheiro com IBE, foi necessário implementar um esquema de criptografia híbrida, desta forma é gerada uma chave simétrica aleatória de 256 bits com a qual é cifrado o ficheiro com *Advanced Encryption Standard* (AES) e por sua vez a chave simétrica é cifrada com IBE utilizando a identidade escolhida.

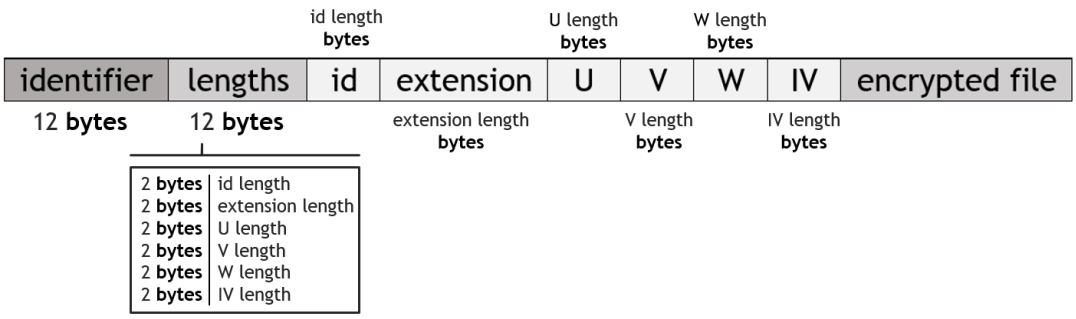


Figura 3.1: Representação de um ficheiro cifrado com o esquema *BF Fullident*

A cifra e decifra envolve elementos algébricos, os quais têm de ser passados com o ficheiro cifrado, para tal houve necessidade de definir um cabeçalho para o ficheiro, que incluísse estes elementos e outra informação necessária no processo de decifra do ficheiro.

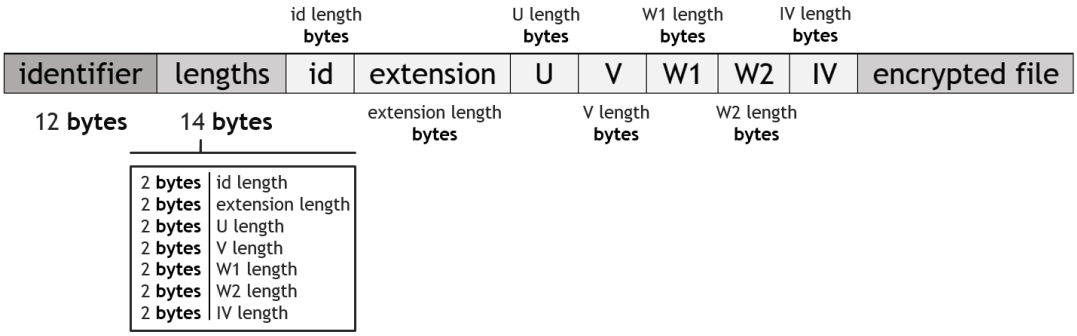


Figura 3.2: Representação de um ficheiro cifrado com o esquema *Emribe*

Na **Figura 3.1** e **3.2** é possível observar uma representação dos ficheiros cifrados (com o respetivo cabeçalho) para os esquemas *BF Fullident* e *Emribe* respetivamente. Assim, para um ficheiro cifrado com o esquema *BF Fullident* temos o campo ***identifier***, que, tal como o nome indica, funciona como identificador do tipo de ficheiro. Neste caso, a designação escolhida foi *"ibeprototype"*, o campo ***lengths*** contém informação sobre os tamanhos dos campos seguintes no cabeçalho, no campo ***id*** está contida a informação sobre a identidade/chave pública com a qual está cifrado o ficheiro, o campo ***extension*** consiste na informação sobre a extensão do ficheiro original, os campos ***U***, ***V***, ***W*** e ***IV*** consistem em informação sobre os elementos necessários para o processo de decifra do ficheiro, por último o campo ***encrypted file*** contém o conteúdo do ficheiro cifrado com o algoritmo AES.

Estas aplicações em *C#* são um passo para a criação da biblioteca para a implementação de sistemas de cifra e decifra de ficheiros baseados em IBE, descrita mais à frente neste capítulo.

3.4 Análise de Eficiência Computacional

Uma das características pretendidas na biblioteca e no sistema de IBE com o CC, foi a funcionalidade de cifra de um ficheiro para múltiplas identidades. Enquanto que o esquema *Emribe* implementa esta funcionalidade por defeito (sendo esta o ponto forte do esquema), no esquema *BF Fullident* é necessário cifrar a mensagem múltiplas vezes, criando uma composição *n*-sequencial do esquema original (*n* como o número de identidades/vezes que a mensagem será cifrada), assim o algoritmo de cifra descrito na **Secção 3.2.1** para *n* identidades será:

$$C = \langle r_i P,\ \sigma \oplus H_2(g_{ID_i}^r),\ M \oplus H_4(\sigma) \rangle \text{ onde } g_{ID_i} = \hat{e}(F_{ID_i}, P_{pub}) \in \mathbb{G}_2^*$$

onde ID_i corresponde a uma identidade e $i \in [1,n]$. Assim para a cifra de uma mensagem com este esquema para *n* identidades, temos *n* operações de *pairing* $\left(\text{para calcular } g_{ID_i} = \hat{e}(F_{ID_i}, P_{pub})\right)$, nenhuma adição, *n* multiplicações no grupo $\mathbb{G}_1$ (para calcular $r_i P$) e *n* exponenciações no grupo $\mathbb{G}_2$ (de forma a calcular $g_{ID_i}^r$).
Para a cifra de uma mensagem para *n* identidades no esquema *Emribe* não é efetuado qualquer *pairing* $\left(\text{o pairing } \hat{e}(Q,P_{pub}) \text{ é pré-calculado durante a fase de } \textbf{Setup} \text{ e utilizado como parâmetro público}\right)$, é efetuada uma exponenciação no grupo $\mathbb{G}_2$ $\left(\text{para calcular } \hat{e}(Q,P_{pub})^r\right)$, são efetuadas $n + 3$ multiplicações no grupo $\mathbb{G}_1$ $\left(rQ,\ rP \text{ e } \hat{e}(Q,P_{pub})^r R \text{ uma vez e } rH_1(ID_i)\ n \text{ vezes}\right)$ e *n* adições no grupo $\mathbb{G}_1$ $\left(\text{para calcular } rH_1(ID_i) + rQ\right)$. Na **Tabela 3.7** é possível observar mais facilmente a comparação do número de operações sobre a curva elíptica destes dois esquemas:

	Pairings	Add. in $\mathbb{G}_1$	Multi. in $\mathbb{G}_1$	Exp. in $\mathbb{G}_2$
BF Fullident	n	0	n	n
Emribe	0	n	n + 3	1

Tabela 3.7: Número de operações na curva elíptica entre os esquemas *BF Fullident* e *Emribe*

A construção apresentada dos esquemas *BF Fullident* e *Emribe* para múltiplas identidades implicam alterações no cabeçalho do ficheiro cifrado, apresentado na secção anterior.

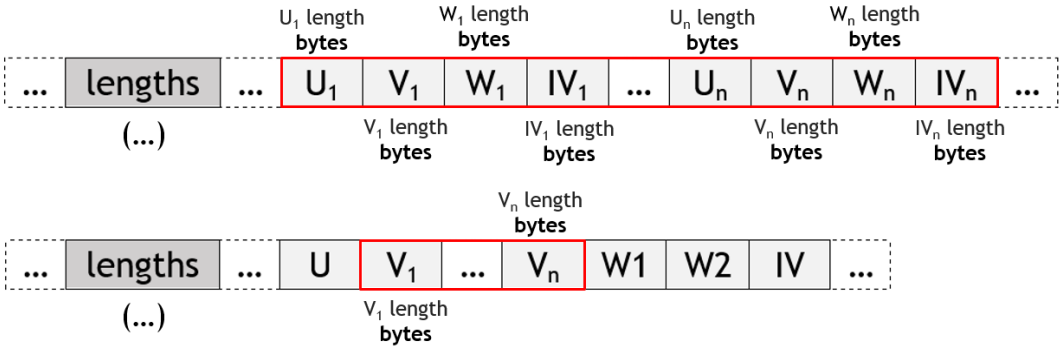


Figura 3.3: Alterações no cabeçalho do ficheiro cifrado com *n* identidades para os esquemas *BF Fullident* e *Emribe* respetivamente

Na **Figura 3.3** é possível observar as alterações (delineadas a vermelho) efetuadas no cabeçalho do ficheiro para conter os novos elementos algébricos necessários à cifra para múltiplas identidades. Em ambos os esquemas existem alterações no campo ***lengths***, no entanto, a diferença de tamanho é mínima (mais 2 bytes para representar o número de identidades com as quais o ficheiro foi cifrado). No caso do esquema *BF Fullident* as alterações no tamanho são notórias, para *n* identidades o tamanho total dos elementos algébricos no cabeçalho dos ficheiros será:

$$\langle n \times (\boldsymbol{U} \text{ length} + \boldsymbol{V} \text{ length} + \boldsymbol{W} \text{ length} + \boldsymbol{IV} \text{ length}) \rangle$$

Para o esquema *Emribe* as alterações são significativamente menores, apenas o elemento $\boldsymbol{V}$ sofre alterações na cifra para múltiplas identidades: $(\boldsymbol{V}_1,\ldots,\boldsymbol{V}_n)$

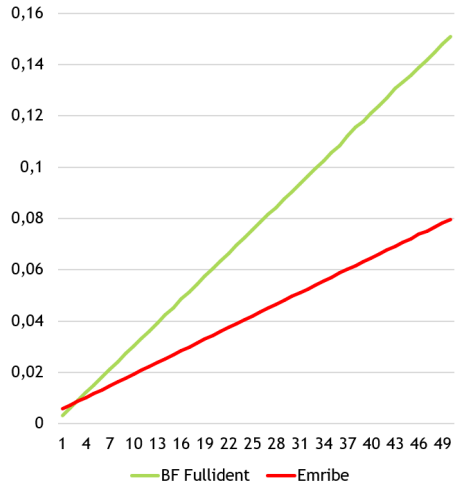
$$\langle \boldsymbol{U} \text{ length} + (n \times \boldsymbol{V} \text{ length}) + \boldsymbol{W} \text{ length} + \boldsymbol{IV} \text{ length} \rangle$$

A **Tabela 3.8** mostra a comparação em termos de tamanho do cabeçalho (é considerado apenas o tamanho dos elementos algébricos, uma vez que o tamanho das restantes informações é idêntico entre os dois esquemas) de um ficheiro cifrado para *n* identidades, com ambos os esquemas e uma chave de curva elíptica com 256 bits.

n-identities	BF Fullident	Emribe
1	240 bytes	464 bytes
2	480 bytes	592 bytes
3	720 bytes	720 bytes
4	960 bytes	848 bytes
5	1200 bytes	976 bytes
10	2400 bytes	1616 bytes

Tabela 3.8: Comparação do tamanho do cabeçalho entre os esquemas *BF Fullident* e *Emribe*

É possível observar em função da tabela que o esquema *Emribe* é mais eficaz em termos de tamanho do cabeçalho. Para este exemplo a adição de uma identidade no esquema *BF Fullident* representa uma adição no tamanho de 242 bytes, contra 130 bytes do esquema *Emribe*.



n-identities	BF Fullident	Emribe
1	0.0031	0.0057
2	0.0061	0.0073
3	0.0090	0.0087
4	0.0122	0.0101
5	0.0151	0.0118
	...	
10	0.0302	0.0193
	...	
25	0.0756	0.0419
	...	
50	0.1508	0.0796

Figura 3.4 & Tabela 3.9: Comparação dos tempos (em segundos) de operações sobre curva elíptica para *n* identidades entre os dois esquemas *BF Fullident* e *Emribe*

Para além das comparações já apresentadas, foram efetuados testes para avaliar a performance real dos dois esquemas na cifra para múltiplas identidades (estes testes foram efetuados numa máquina com processador *Intel i7-3610QM, 2.30Ghz*), com a mesma curva elíptica.

A tabela e o gráfico da **Figura 3.4 & Tabela 3.9** apresentam o resultado dos testes realizados, o esquema *BF Fullident* é ligeiramente mais eficiente para cifra com uma e duas identidades, para mais de duas identidades o esquema *Emribe* é superior.

Tendo em conta todas as comparações efetuadas, o esquema escolhido para a utilização na biblioteca para cifra e decifra de ficheiros em *.NET C#* foi o esquema *Emribe*. Esta escolha é justificada pela eficiência computacional (superior ao esquema *BF Fullident*) no tempo de cifra e pelo tamanho reduzido necessário no cabeçalho do ficheiro cifrado resultante.

3.5 Biblioteca *.NET C#*

A biblioteca criada para *.NET C#* usa o *wrapper* desenvolvido para o esquema *Emribe* e permite implementá-lo de uma forma simples numa aplicação ou 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.

Funcionalidades de autenticação, para geração de uma chave privada junto de um PKG, não estão implementadas na biblioteca. As funções da biblioteca podem ser divididas em dois grupos, as que serão para utilização pelo PKG e as que serão para utilização pelo cliente/utilizador do sistema.

3.5.1 Funções do PKG

O PKG têm como função a geração dos 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 a identidade *id*. 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)

Estas funções recebem como parâmetro uma *string* com a descrição da curva elíptica (selecionada pelo PKG) e só necessitam de ser executadas uma vez. Após a geração destes parâmetros, a chave privada *s* têm de permanecer do conhecimento exclusivo do PKG (a segurança do sistema depende desta exclusividade) e os parâmetros públicos e a definição da curva elíptica utilizada devem ser disponibilizados aos utilizadores do sistema.

Para a geração de uma chave privada, devem ser passados como parâmetros à função a identidade (*id*), a curva elíptica e chave privada do sistema:

byte[] GetPrivateKey(string id, string curve, byte[] masterKey)

3.5.2 Funções do Cliente/Utilizador

O utilizador têm ao seu dispor funções para a cifra e decifra de ficheiros. Estas permitem que o utilizador efetue a cifra e decifra de ficheiro de forma simples e sem necessidade que este conheça o funcionamento interno do algoritmo de cifra:

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)

A função de cifra do ficheiro recebe como parâmetros a(s) identidade(s) com a(s) qual/quais o utilizador pretende cifrar o ficheiro, uma string opcional com informação adicional para adicionar às identidades/chave públicas (exemplo: data de validade da chave privada), o ficheiro a ser cifrado, string opcional com o nome ou extensão original do ficheiro a cifrar, definição da curva elíptica e parâmetros públicos *P*, *Q* e $\hat{e}(Q,P_{pub})$.
Esta função implementa ainda o elemento σ descrito na **Secção 3.2.2**, desta forma é possível verificar a integridade do cabeçalho durante a função de decifra.
Na função de decifra do ficheiro, esta recebe a identidade e a chave privada correspondente, o ficheiro cifrado, a curva elíptica e o parâmetro público P_{pub}.

Caso exista algum problema nos parâmetros passados às funções, como exemplo, ficheiro inválido, erro ao verificar a integridade ou chave privada errada, a biblioteca lança a exceção adequada para informar o utilizador do que está errado.

Para além das funções apresentadas a biblioteca fornece também ao utilizador funções auxiliares para recuperar informação do ficheiro cifrado, como o nome do ficheiro original (caso este exista uma vez que este parâmetro é opcional na função de cifra), recuperar a informação adicional adicionada às identidades/chave pública (caso esta exista, na arquitetura desenvolvida seria o *hash* da políticas, o qual irá ser falado no seguinte capítulo) e recuperar a(s) identidade(s) com a(s) qual/quais foi cifrado o ficheiro:

string GetOriginalFilename(MemoryStream streamIn)
string GetPolicy(MemoryStream streamIn)
string[] GetIdentities(MemoryStream streamIn)

No **Anexo A.4** pode-se observar o código fonte de um exemplo da utilização da biblioteca, onde são utilizadas as funções do PKG e do Cliente/Utilizador. É efetuada a geração de parâmetros, a cifra de um ficheiro com duas identidades diferentes e a respetiva decifra com a chave privada gerada correspondente a uma das identidades utilizadas.

3.6 Conclusão

Neste capítulo foram apresentadas as descrições técnicas dos esquemas de IBE e IBS implementados. Estes foram implementados em linguagem *C* numa primeira fase, sob a forma de pequenas aplicações de teste com funcionalidades simples de cifra e decifra e de assinatura. Numa segunda fase os três esquemas foram implementados em *wrappers* para a utilização na linguagem *.NET C#*, de forma a possibilitar o desenvolvimento nesta linguagem. As funções dos *wrappers* foram apresentadas e foi feita a correspondência entre estas funções e as várias operações no algoritmo dos esquemas.

Foram desenvolvidas duas aplicações para cifra e decifra em *.NET C#*, baseadas nos esquemas de IBE e explicadas as implementações que foram necessárias para o seu funcionamento correto. Nomeadamente a implementação de um esquema de criptografia híbrida em que o algoritmo AES é utilizado juntamente com IBE e a definição dos cabeçalhos dos ficheiros cifrados. A nível da análise de eficiência computacional foram feitos estudos comparativos entre os dois esquemas implementados (*BF Fullident* e *Emribe*) quando utilizados para cifrar com múltiplas identidades. Para tal foram analisadas o número de operações sobre a curva elíptica, o tamanho dos cabeçalhos dos ficheiros resultantes da cifra e a comparação dos tempos das operações sobre curva elíptica referidas.

Após a análise de eficiência dos esquemas de IBE foi escolhido e implementado o esquema *Emribe* na criação de uma biblioteca para *.NET C#*, com funções para o desenvolvimento de um sistema completo de cifra e decifra de ficheiros baseado em identidade.

O objetivo deste capítulo prende-se à apresentação dos esquemas utilizados, das suas implementações e dos desafios e opções tomadas durante o desenvolvimento. Procura-se desta forma

fornecer um visão aprofundada dos algoritmos e esquemas subjacentes utilizados na arquitetura especificada no capítulo seguinte.

Capítulo 4

Arquitetura

4.1 Introdução

No **Capítulo 3** foram apresentados os esquemas criptográficos nos quais se baseia a arquitetura desenvolvida. Neste capítulo pretende-se abordar a arquitetura de cifra e decifra de ficheiros e o sistema de associação de políticas de privacidade (*sticky policies*[1]) durante o processo de cifra dos ficheiros. É apresentada a estrutura da arquitetura, bem como os diferentes componentes que a compõem de uma forma pormenorizada. Pretende-se justificar as soluções implementadas e fornecer uma visão do funcionamento global, assim como de cada uma das fases.

4.2 Cenários de Utilização

Relativamente aos cenários de utilização da arquitetura para a cifra e decifra de ficheiros com IBE e recuperação de chaves utilizando o CC, foram considerados três cenários distintos, e alguns exemplos de como poderiam beneficiar com um sistema desta natureza.

Hospital/Clínica. Este cenário é semelhante à infraestrutura criada por investigadores da HP e apresentada na **Secção 2.5.1** para um sistema de informação para confidencialidade e privacidade de mensagens num ambiente hospitalar. Apresentamos alguns exemplos:

- A informação médica de um paciente apenas poderá ser acedida/decifrada pelo(s) seu(s) médico(s) ou por ele próprio;
- Em caso de emergência deve ser possível decifrar as informações com autorização de uma entidade superior (por exemplo, direção do hospital). Este caso torna a custódia de chaves (key escrow), referida no **Capítulo 2** como uma desvantagem, numa vantagem particularmente útil neste tipo de situações.

Empresa. Para este cenário apontamos como exemplos de utilização:

- Informações importantes podem ser cifradas com múltiplas identidades em associação com o cargo desempenhado dentro da empresa, assim um utilizador pode perder as permissões para decifrar esses dados se despromovido;
- A cifra de informações que só devem ser disponibilizadas numa certa data, assim será impossível obter uma chave privada antes da data estipulada para decifrar as informações.

Escola/Universidade. Neste cenário apontamos como exemplos de utilização:

[1] O conceito de ***sticky policies*** consiste na associação de políticas a dados/informação, no caso da arquitetura criada estas políticas são aplicadas a ficheiros cifrados de forma a controlar e limitar o acesso ao conteúdo do ficheiro original respetivo.

- A cifra das soluções de um teste/exame com a identidade dos alunos que vão efetuar a prova, à qual é associada uma data inicio de validade da chave privada, assim a solução poderia ser colocada online antes da realização da prova, mas só seria possível obter a chave respetiva após a sua realização;
- O exemplo anterior, aplica-se igualmente à disponibilização de testes/exames e outros conteúdos, os alunos podem descarregar as provas, mas só será possível obter uma chave privada para decifrar o ficheiro na data designada pelo professor;

4.3 Descrição do Sistema

A sistema criado foi desenhado com base nas características e funcionalidades pretendidas no início do desenvolvimento do projeto e dos requisitos inerentes dos cenários de uso apresentados na secção anterior. O objetivo principal foi a criação de um sistema de cifra e decifra de ficheiros que fizesse uso do mecanismo de autenticação do CC e das funcionalidades características de um sistema criptográfico baseado em identidade.

4.3.1 Funcionamento Geral

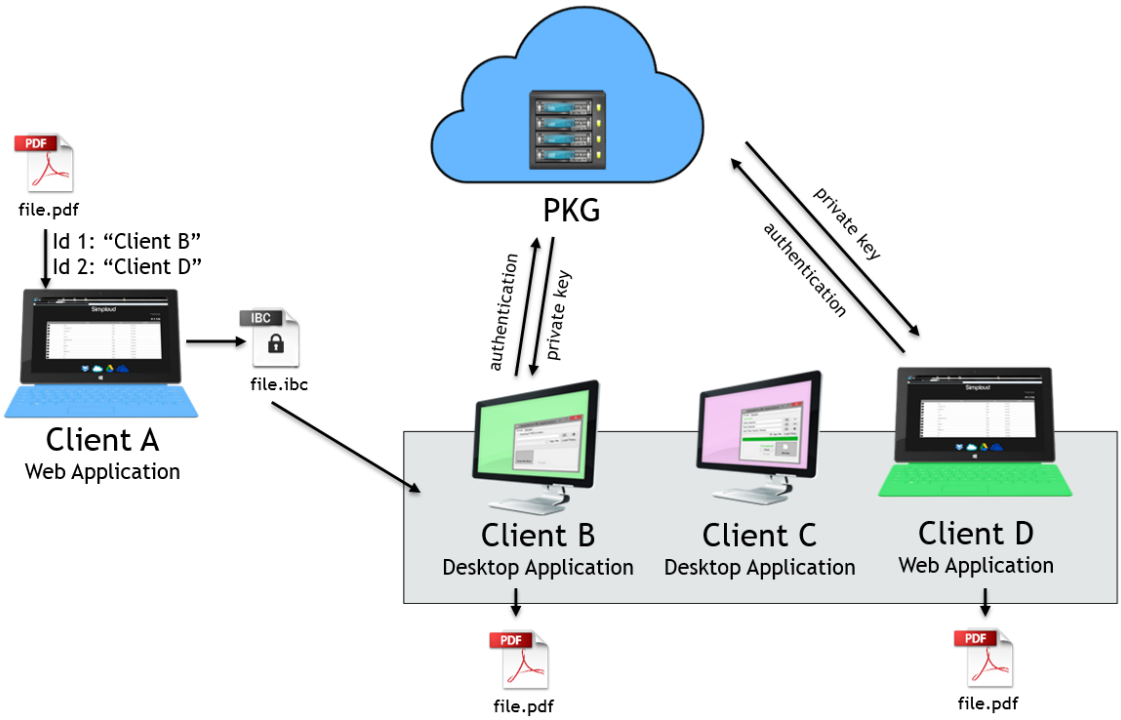


Figura 4.1: Esquema geral da arquitetura criada

Na **Figura 4.1** podemos observar um exemplo de funcionamento da arquitetura criada, temos quatro utilizadores, *A* e *D* utilizam uma aplicação web e os utilizadores *B* e *C*, a aplicação para Desktop. O utilizador *A* cifra o ficheiro "file.pdf"com a identidade dos utilizadores *B* e *D*, e envia o ficheiro cifrado "file.ibc"para uma unidade de armazenamento partilhada pelos utilizadores *B*, *C* e *D*. Os três têm acesso mas apenas os utilizadores *B* e *D* podem autenticar-se ao PKG e recuperar as chaves privadas correspondentes para decifrar e obter o ficheiro original.

4.3.2 Caracteristica de Identidade

A utilização de um sistema criptográfico baseado em identidade permite-nos selecionar qualquer cadeia de caracteres (*string*), que identifique inequivocamente um utilizador, para utilização como chave pública. Na arquitetura implementada e descrita neste capítulo, o CC é o componente essencial para a autenticação do utilizador junto do PKG. Assim, e sem aprofundar demasiado no processo de autenticação (descrito em detalhe mais à frente neste capítulo), é importante que a característica/*string* utilizada como chave pública possa ser confirmada com o certificado público do utilizador contido no cartão.

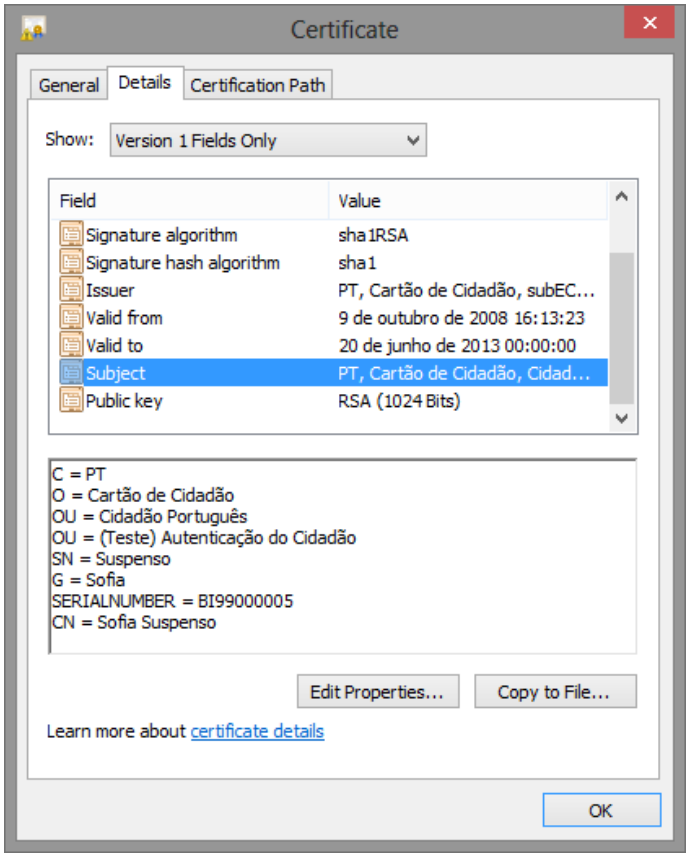


Figura 4.2: Exemplo da informação contida no campo *subject* do certificado público de autenticação contido no CC

Na **Figura 4.2** podemos observar a informação apresentada no campo *subject* do certificado público de autenticação do CC, onde se destaca o nome completo do utilizador e o seu número de identificação civil.

A característica de identidade escolhida para usar como chave pública nesta arquitetura foi o nome completo do utilizador. Apesar desta escolha não ser ideal, uma vez que não é uma característica que identifique de forma inequívoca um utilizador (podem existir utilizadores com o mesmo nome) é suficiente numa fase de protótipo para mostrar as capacidades de um sistema desta natureza.
O nome poderia ser substituído pelo número de identificação civil (número do CC), e assim teríamos uma característica inequívoca da identidade de um utilizador. Contudo surge outro problema, o utilizador necessita de saber o número de identificação das pessoas para as quais pretende cifrar o ficheiro. Este problema pode ser solucionado de várias formas, dependendo

do contexto de utilização do sistema.

Para um sistema utilizado por uma empresa, uma das soluções para este problema passa por utilizar o nome completo, ou o primeiro e último nome em conjunto com uma característica que assim identificasse o utilizador dentro da empresa (características, consideradas à partida, como conhecidas pelo utilizador remetente). Por exemplo, utilizando o nome e cargo da pessoa nessa empresa, para tal seria necessário a ligação de uma base de dados ao PKG com informação sobre os empregados (nome, número do CC e informação sobre o cargo).
Num sistema para uma universidade a solução poderia passar por utilizar o número de estudante, docente e funcionário como caracteristica de identidade.

Ambas a soluções apresentadas necessitariam de um registo prévio do utilizador num base de dados, contudo neste tipo de contextos é de esperar que este registo exista assim que o utilizador passe a fazer parte da organização ou universidade.

Outra solução possível, seria utilizar todo o certificado de autenticação como chave pública. Esta solução implicaria no entanto a existência de um repositório ou do envio prévio deste certificado do destinatário para o recetor, assim como a perda da característica principal de um sistema IBC.

4.3.3 Políticas

O conceito de *sticky policies* foi introduzido em 2002 por Karjoth *et al.* [KSW02], a ideia na qual este conceito se baseia é a associação de um conjunto de políticas e regras a informações de forma a tornar possível o controlo de acesso a estas informações. Uma arquitetura típica de *sticky policies* é composta por três elementos, o dono da informação, o recetor e uma autoridade de confiança responsável pela confirmação e aplicação das políticas.

De forma a facilitar a definição e compreensão de políticas pelos utilizadores estas são normalmente definidas utilizando linguagens próprias, tais como *Enterprise Privacy Authorization Language* [AHK+03] (EPAL), *eXtensible Access Control Markup Language* [GM03] (XACML) e *eXtensible Rights Markup Language* [WLD+02] (XrML). Estes três exemplos bem como a maior das linguagens de definição de políticas têm como base a linguagem *Extensible Markup Language* (XML).

No contexto da arquitetura descrita, e uma vez que o pequeno número de políticas sem grande complexidade não justificava a implementação de uma das linguagens anteriores, optou-se por criar de raiz uma especificação simples de políticas baseada igualmente na linguagem XML que permitisse tornar tanto a criação, como a compreensão de políticas, fácil e transparente para o utilizador. Contudo, a forma como a verificação e o controlo de políticas está implementado permite que seja possível alterar a linguagem proposta a qualquer momento para uma das apresentadas ou para outra linguagem de políticas não referida.

A implementação deste sistema de políticas de ficheiros na arquitetura é em parte semelhante à descrita por Mont *et al.* [MPB03]. O(s) utilizador(es) a quem os ficheiros foram destinados (cifrados com a identidade deste(s) utilizador(es)) têm que respeitar as políticas associadas de forma a obter uma chave privada capaz de decifrar os ficheiros e obter o conteúdo original.

No âmbito da arquitetura criada foram definidas duas políticas, como exemplo, uma regra temporal, designada pelo elemento ***DATE*** e uma regra relacionada com o nível de segurança do utilizador à qual nos podemos referir como segurança baseada em papeis (*role based security*), ***SECURITY***.
O elemento ***DATE*** têm dois parâmetros, ***Start*** e ***End***, e consiste numa regra para definir o período temporal na qual uma chave privada pode ser gerada pelo PKG. Assim, o utilizador pode definir três modos para controlar a validade da chave privada:

- ***End***: o utilizador pode obter a chave privada até à data definida no parâmetro ***End***, depois desta data é impossível obter a chave;
- ***Start***: o utilizador pode obter a chave privada depois da data definida no parâmetro ***Start***, antes desta data é impossível obter a chave e decifrar o ficheiro;
- ***Start*** e ***End***: o utilizador pode obter a chave privada no período correspondente entre a data definida no parâmetro ***Start*** e a data definida no parâmetro ***End***.

Para o elemento ***SECURITY*** é necessária uma base de dados no PKG ou a que este possa ter acesso. A ideia desta regra é uma entidade superior, poder definir vários níveis de segurança consoante o papel que o utilizador tem dentro da organização que utiliza esta arquitetura. Foram definidas quatro *strings* que pretendem representar níveis crescentes de segurança para exemplificação, ***None*** (sem nível definido), ***Level 1***, ***Level 2*** e ***Level 3***. Um utilizador com nível superior tem acesso aos níveis de segurança inferiores, por exemplo, se o utilizador tiver nível de segurança, ***Level 2***, tem acesso aos níveis, ***Level 2***, ***Level 1*** e ***None***, mas nunca a níveis superiores. Esta regra requer que o PKG tenha acesso a uma lista com os utilizadores do sistema e o respetivo nível de segurança.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <POLICY>
3    <DATE active="true">
4      <Start>11/01/2013-00:00</Start>
5      <End>01/01/2014-00:00</End>
6    </DATE>
7    <SECURITY>
8      <Type>Level 1</Type>
9    </SECURITY>
10 </POLICY>
```

Figura 4.3: Exemplo de um ficheiro XML de políticas

Na **Figura 4.3** é possível observar um exemplo de um ficheiro XML de políticas para esta arquitetura. De forma a obter a chave privada para decifrar um ficheiro, cifrado com a identidade **X** e a com as políticas descritas neste exemplo, é necessário cumprir as duas regras. A chave só poderá ser obtida entre 1 de novembro de 2013 e 1 de Janeiro de 2014 e apenas se o utilizador com a identidade **X** tiver nível de segurança ***Level 1*** na organização à qual pertence, se este nível for alterado pela organização para **None**, o utilizador deixa de poder obter a chave privada.

Para além das duas políticas apresentadas foram ainda contemplada a utilização do endereço de IP, em que a ideia é permitir que o utilizador apenas possa obter a chave privada para decifrar um ficheiro se o processo de autenticação for iniciado a partir de uma gama de IPs específica.

A criação destas políticas é realizada pelo utilizador, mas a confirmação e aplicação das mesmas, é efetuada pelo serviço de PKG no processo de autenticação e geração de chaves privadas, como será mostrado mais à frente nesta secção.

4.3.4 Cifra e Assinatura

O processo de cifra de ficheiros para múltiplos utilizadores, utilizado na arquitetura, tem como base o esquema *Emribe* e a sua implementação foi efetuada com recurso à biblioteca descrita no **Capítulo 3**. Para além da cifra de ficheiros descrita no capítulo anterior, o utilizador tem ainda as opções de assinar o ficheiro com a sua identidade (esquema *Paterson IBS*), e associar informação sobre um ficheiro com uma descrição de políticas XML à(s) chave(s) pública(s).

Se o utilizador pretender apenas cifrar o ficheiro, não é necessário qualquer contato com o PKG, desde que a aplicação disponha dos respetivos parâmetros públicos e da definição de curva elíptica. Contudo, se pretender assinar o ficheiro, é necessário autenticar-se no PKG através da aplicação com o CC e obter a chave privada correspondente à sua identidade (ver **Secção 4.3.6**). No caso da assinatura, o cabeçalho do ficheiro cifrado irá sofrer alterações em relação ao apresentado para o esquema *Emribe* no capítulo anterior, de forma a poder guardar todas as informações e elementos necessários para a verificação da assinatura.

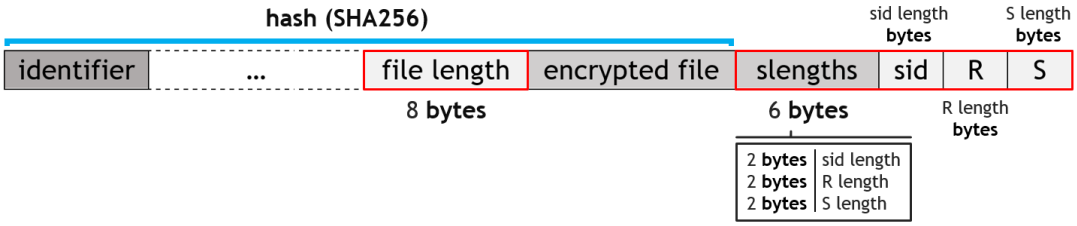


Figura 4.4: Cabeçalho de um ficheiro cifrado e assinado

Na **Figura 4.4** é possível observar a azul a parte correspondente dos elementos do ficheiro para a qual é efetuado o *hash*, este será assinado com a chave privada do utilizador. A vermelho está assinalada a adição dos campo ***file length*** de 8 bytes, que contém o tamanho do ficheiro original e a adição dos campos ***slengths***, ***sid***, ***R*** e ***S***. O campo ***slengths*** contém informação sobre o tamanho dos elementos seguintes, ***sid*** contém a identidade do utilizador que assinou a mensagem e os campos ***R*** e ***S*** correspondem aos elementos algébricos necessários para a verificação da assinatura. O campo ***file length*** é necessário de forma a calcular o tamanho do ficheiro cifrado[2] durante o processo de decifra e saber onde acaba o ficheiro e começam os parâmetros de assinatura. Os últimos quatro campos são exclusivos da assinatura e só constam no ficheiro se o utilizador decidir cifrar e assinar.

Caso o utilizador decida associar um ficheiro com políticas à(s) chave(s) pública(s), esta associação é realizada combinando a(s) identidade(s) com o *hash* do respetivo ficheiro de políticas, assim a chave pública será ***identidade + hash ficheiro XML***. Esta utilização do *hash* na chave pública, vai criar a ligação entre as políticas e o ficheiro cifrado, uma vez que *hash* utilizado só poderá ser gerado com as políticas originais. Se um atacante tentar alterar as políticas, a

[2]Uma vez que o ficheiro é cifrado por blocos de 128 bits com o algoritmo *AES* com chave de 256 bits, o tamanho do ficheiro cifrado em bytes pode ser calculado através da fórmula *ciphersize* = (*filesize*/16 + 1) $\times$ 16

chave privada gerada não será capaz de decifrar o ficheiro. As políticas terão que ser enviadas juntamente com o ficheiro resultante da cifra.

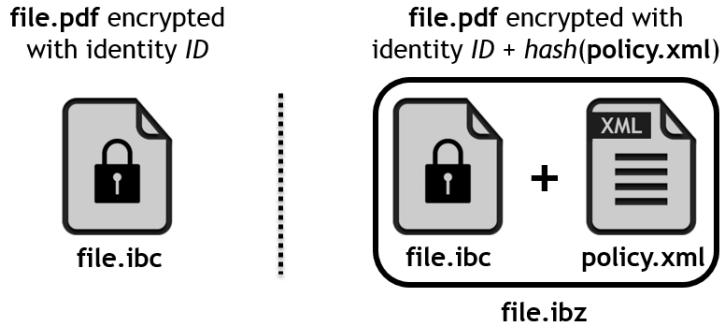


Figura 4.5: Ficheiro *file.pdf* cifrado com a identidade *ID* e com *ID* + *Políticas*

Após o processo de cifra, o ficheiro cifrado resultante terá extensão *.ibc*, desta forma um utilizador pode distinguir de forma fácil um ficheiro cifrado com esta arquitetura. Se para além da cifra, o ficheiro for associado a um conjunto do políticas, a aplicação irá criar um contentor com o XML das políticas e com o ficheiro cifrado (*.ibc*), este contentor terá a extensão *.ibz* e consistirá num ficheiro ZIP (ver **Figura 4.5**). A utilização deste tipo de ficheiro têm dois objetivos, transportar a definição das políticas junto do ficheiro cifrado e facilitar o acesso do utilizador às políticas de forma a poder visualizar facilmente o seu conteúdo. É possível retirar tanto o ficheiro de políticas, como o ficheiro cifrado do contentor *.ibz*, no entanto, nunca será possível obter a chave privada junto do PKG, se este não tiver acesso às políticas associadas (de forma a verificar o cumprimento ou não das mesmas) na cifra do ficheiro original.

4.3.5 Decifra

Tal como na cifra, a decifra é baseada na biblioteca descrita no **Capítulo 3**. No processo de decifra, a aplicação pode receber os dois tipos de ficheiros resultantes mencionados na secção anterior (*.ibc* e *.ibz*). Se o tipo de ficheiro for *.ibz* a aplicação vai extrair as políticas XML e o ficheiro cifrado *.ibc*.
Caso o ficheiro *.ibc* tenha sido assinado, é calculado o *hash* correspondente ao cabeçalho e ao conteúdo cifrado, e verificada a assinatura com os elementos presentes no final do ficheiro. O utilizador é informado pela aplicação do resultado da verificação de assinatura e pode decidir se pretende decifrar ou não o ficheiro.
Para decifrar o ficheiro, o utilizador necessita de se autenticar corretamente através da aplicação com o seu CC no PKG e obter a chave privada correspondente à sua identidade ou identidade mais o *hash* das políticas.

4.3.6 Comunicação, Autenticação e Geração de Chaves Privadas

O processo de autenticação do utilizador junto do serviço de PKG é apenas necessário para a geração de chaves privadas. Durante este processo, o CC do utilizador (nomeadamente o certificado de autenticação do CC) é utilizado como forma de comprovar a identidade de um utilizador. A chave privada de um utilizador pode ser correspondente apenas à sua identidade ou à identidade em junção com as políticas descritas anteriormente.
Durante o processo de comunicação, a aplicação e o PKG comunicam entre si utilizando sessões sobre *Transport Layer Security/Secure Sockets Layer* (TLS/SSL). Uma sessão é iniciada assim que

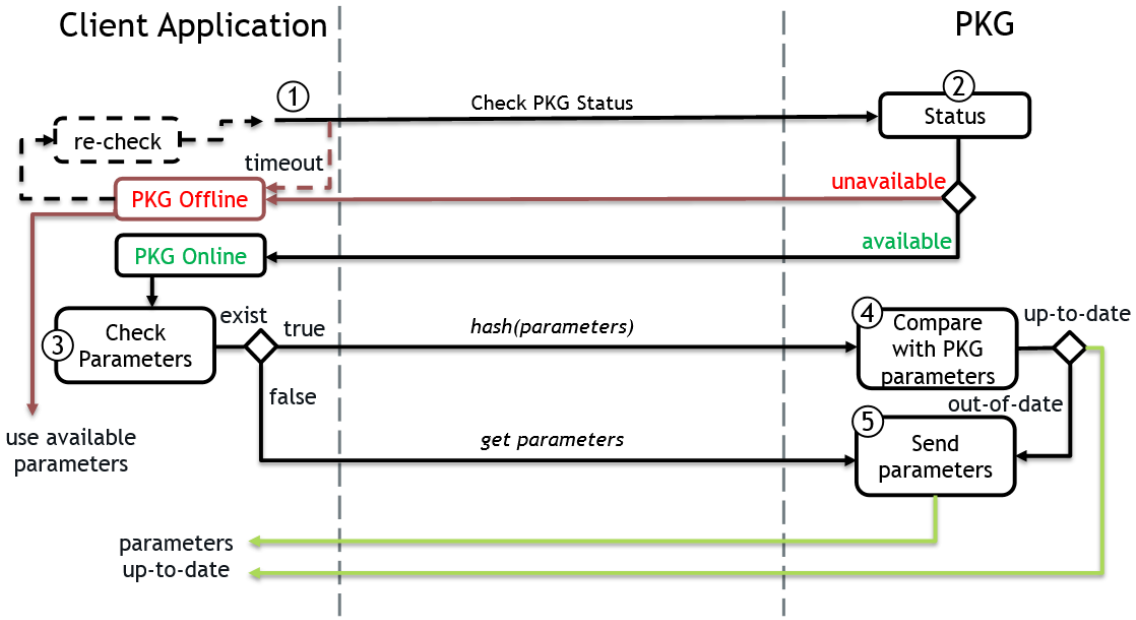


Figura 4.6: Processo de verificação e obtenção de parâmetros e definição de curva elíptica

o utilizador executa a aplicação e esta consegue estabelecer uma ligação com o PKG. A **Figura 4.6** representa o processo de obtenção de parâmetros públicos e definição de chave elíptica:

1. A aplicação contacta o PKG de forma a obter o seu estado, caso este pedido atinja um período de *timeout*, a aplicação vai assumir que o PKG está offline e irá utilizar os parâmetros e a definição de curva disponíveis (caso os tenha, caso contrário não será possível cifrar ficheiros), o utilizador pode a qualquer momento verificar o estado do PKG de forma a tentar estabelecer uma ligação;

2. Caso seja estabelecida uma ligação, o PKG informa a aplicação se está disponível ou indisponível (o serviço de PKG pode estar desativado, por exemplo para manutenção);

3. Se o PKG estiver disponível a aplicação vai verificar se tem parâmetros públicos e definição de curva elíptica, se os tiver, vai enviar o *hash* destes para o PKG, caso contrário vai pedir estes elementos ao PKG;

4. O PKG ao receber o *hash* dos elementos da aplicação vai compará-lo com o *hash* dos elementos que possui, se este coincidir é enviada a informação à aplicação que os elementos estão atualizados;

5. Caso a aplicação não disponha dos parâmetros públicos e definição de curva elíptica ou caso estejam desatualizados, estes serão enviados pelo PKG.

A sessão de um utilizador termina quando este recebe a chave privada, uma mensagem de erro (caso não seja possível gerar a chave pretendida devido ao incumprimento das políticas ou caso o utilizador não seja capaz de se autenticar corretamente) ou quando a sessão atinge o tempo máximo de inatividade.
Na **Figura 4.7** é possível visualizar o processo de autenticação de forma a obter uma chave privada junto do PKG:

1. A aplicação envia o certificado de autenticação do CC para o PKG, a identidade do utilizador, o ficheiro com a descrição das políticas e o *hash* das políticas (a descrição e o *hash*

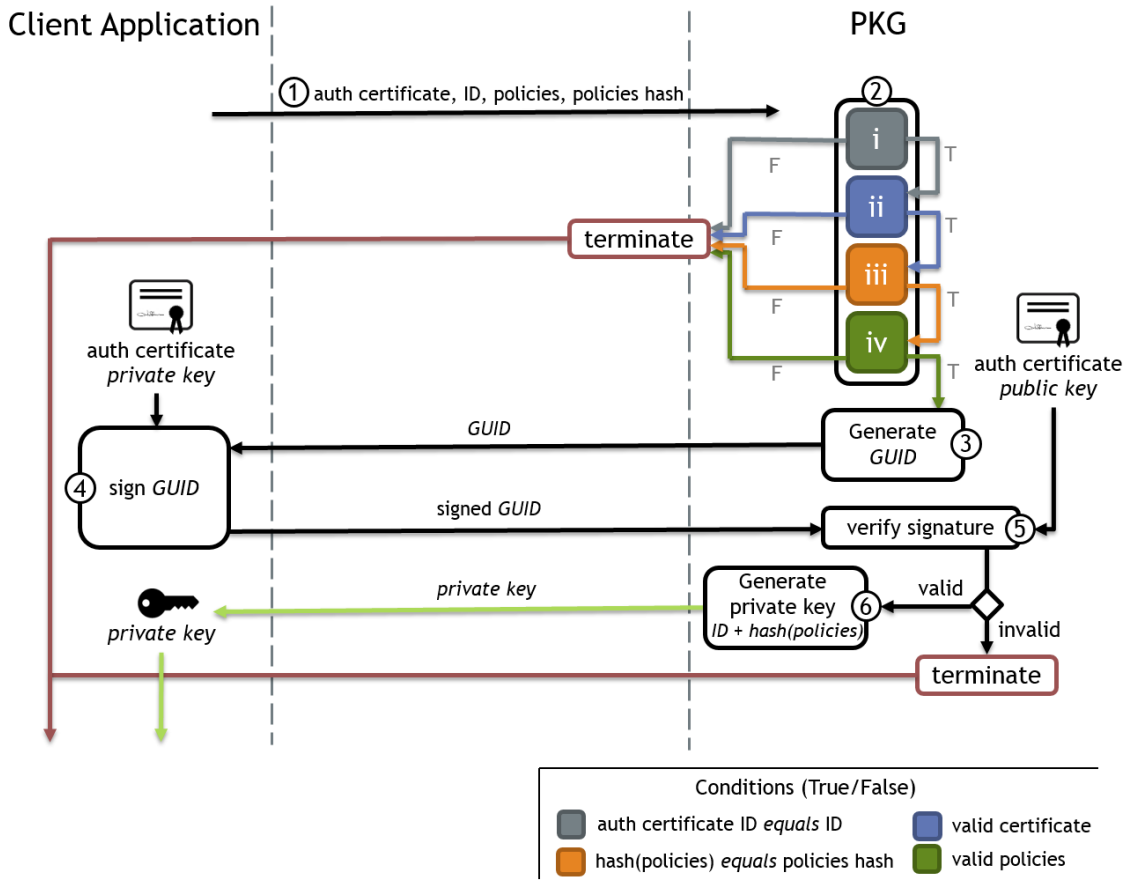


Figura 4.7: Processo de autenticação e obtenção de chave privada junto do PKG

são opcionais, uma vez que um ficheiro pode ser cifrado apenas com a identidade) que foi utilizado em conjunto com a(s) identidade(s) para cifrar um ficheiro;

2. O PKG ao receber os dados enviados pelo utilizador irá efetuar verificações de forma a validar estes dados:

   i. Confirmar que a identidade transmitida corresponde à identidade presente no certificado de autenticação;

   ii. Verificar se o certificado enviado é realmente um certificado de autenticação do CC, se é válido (através da construção da uma cadeia de certificados) e se não foi revogado;

   iii. Calcular e confirmar se o *hash* da descrição das políticas é igual ao *hash* enviado pela aplicação;

   iv. Verificar a construção das políticas e confirmar a sua aplicação.

   Se qualquer uma destas quatro condições for falsa ou não se verificar, a sessão é imediatamente terminada pelo PKG e a aplicação recebe uma mensagem, indicando que o processo foi terminado e a razão respetiva. No caso do ficheiro cifrado não ter políticas associadas, os processos *iii.* e *iv.* são ignorados e o PKG verifica apenas os dois primeiros processos;

3. Caso todas as verificações sejam efetuadas com sucesso, o PKG irá gerar um *Globally Unique Identifier* (GUID) de 128 bits que será enviado à aplicação;

4. O GUID, após ser recebido pela aplicação, vai ser assinado com a chave privada do certificado de autenticação (é pedido que o utilizador introduza o PIN de autenticação do CC) e enviado de volta para o PKG já assinado;

5. O PKG verifica se a assinatura é válida com a chave pública do certificado de autenticação enviado;

6. Se a assinatura for válida, o PKG vai gerar e enviar a chave privada solicitada pelo utilizador à aplicação. A sessão é terminada pelo PKG após esta operação.

O PKG não guarda qualquer informação de forma permanente sobre os utilizadores, nem armazena as suas chaves. Todos os dados transmitidos pelo utilizador através da aplicação são eliminados no PKG, assim que a sessão é terminada.

4.4 Modelo de Segurança

No **Capítulo 1**, listamos como requisitos básicos de segurança que um sistema criptográfico deve fornecer são, **Confidencialidade**, **Integridade**, **Autenticação** e **Não-Repudiação**.
A segurança do sistema como um todo, irá depender de cada um dos seus elementos. Assim, o sistema irá depender da segurança dos esquemas de IBE e de IBS, nos quais se baseia a cifra de chaves e a assinatura de ficheiros, na segurança do algoritmo AES com chave de 256 bits utilizado na cifra dos ficheiros, no CC como *smartcard*, nos canais de comunicação utilizados e na resistência do PKG a ataques que possam comprometer a sua integridade e a chave privada do sistema.

A integridade de um ficheiro cifrado é garantida em duas fases, na verificação da assinatura do ficheiro, caso este a contenha, e no processo de decifra do ficheiro em que são utilizados os elementos presentes no cabeçalho e o conteúdo decifrado de forma a verificar se a integridade do ficheiro original foi mantida.

É importante num sistema desta natureza, que seja impossível para um atacante fazer-se passar por um utilizador legítimo do sistema ou pelo PKG. Pressupõe-se que os canais de comunicação entre a aplicação e o PKG são seguros. Pressupõe-se igualmente que os esquemas criptográficos subjacentes utilizados são seguros e que o serviço de PKG é um serviço no qual o utilizador pode confiar, e no qual é impossível para um atacante obter a chave privada do sistema.
Em relação ao CC, pressupõe-se que um atacante não pode ler nem modificar as chaves criptográficas armazenadas no *smartcard* ultrapassando as suas resistências à adulteração. Pressupõe-se igualmente que não é possível forjar a assinatura digital do emissor de identidade para modificar um certificado.

4.5 Conclusão

Neste capítulo foi apresentada a arquitetura desenvolvida no decorrer desta dissertação, foi abordada a estrutura geral, bem como a estrutura dos diferentes componentes que a compõem. Foram apresentados três cenários de utilização desta arquitetura que pensamos poderem beneficiar com um sistema desta natureza e fornecidos vários exemplos de utilização que justificam

esta escolha.

Foram apresentados os componentes e as decisões efetuadas, como a escolha da caracteristica de identidade e as suas justificações. Foi apresentado o modelo de políticas XML especificadas e a sua integração e aplicação dentro da arquitetura, bem como cada uma das regras implementadas. A nível da cifra e assinatura foram apresentadas as alterações necessárias no cabeçalho de um ficheiro cifrado e a decisão tomada para a integração do XML de políticas com o ficheiro cifrado. Na decifra foram descritos o processo de decifra em si e a verificação de assinatura. Foram também expostos detalhadamente os processos de comunicação, autenticação e geração de chaves privadas. Foi ainda apresentado o modelo de segurança, baseado na segurança dos elementos que constituem o sistema como um todo.

O objetivo deste capítulo prende-se à apresentação do funcionamento da arquitetura e à justificação das soluções implementadas nas várias fases que constituem o funcionamento global desta.

Capítulo 5

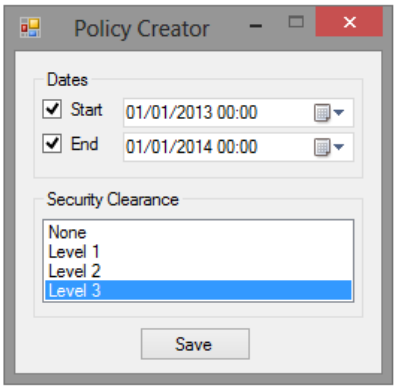
Resultados

5.1 Introdução

Neste capítulo serão apresentados os resultados do sistema implementado. Será apresentada a aplicação para *Windows* de cifra e decifra de ficheiros e uma solução para serviços de armazenamento em *Cloud*, criada no âmbito de uma dissertação desenvolvida paralelamente no Projeto *PRICE*, que utiliza a arquitetura proposta. Para além destas aplicações, será apresentado igualmente o serviço de PKG, que se encontra a correr online na plataforma *Azure*.

5.2 Aplicação para a criação de políticas

As políticas de ficheiros apresentadas no capítulo anterior, podem ser definidas e criadas em linguagem XML, num simples editor de texto pelo utilizador. De modo a facilitar a definição e criação destas, foi criada uma aplicação para *Windows*, em que o utilizador pode especificar de forma simples as políticas desejadas e guardar automaticamente o ficheiro XML, gerado com as especificações selecionadas (ver **Figura 5.1**).




```
1  <?xml version="1.0" encoding="utf-8"?>
2  <POLICY>
3    <DATE active="true">
4      <Start>01/01/2013-00:00</Start>
5      <End>01/01/2014-00:00</End>
6    </DATE>
7    <SECURITY>
8      <Type>Level 3</Type>
9    </SECURITY>
10 </POLICY>
```

Figura 5.1: Definição de políticas e o ficheiro XML resultante

5.3 Aplicação *Windows Desktop*

A aplicação foi criada em *.NET C#* para o sistema operativo *Windows*, para fazer uso de todas as funcionalidades da aplicação, o utilizador necessita de uma ligação à Internet (para autenticação e requisição de chaves junto do PKG), de um leitor de cartões capaz de ler o CC e do *middleware* do CC instalado no computador (para o mecanismo de autenticação).

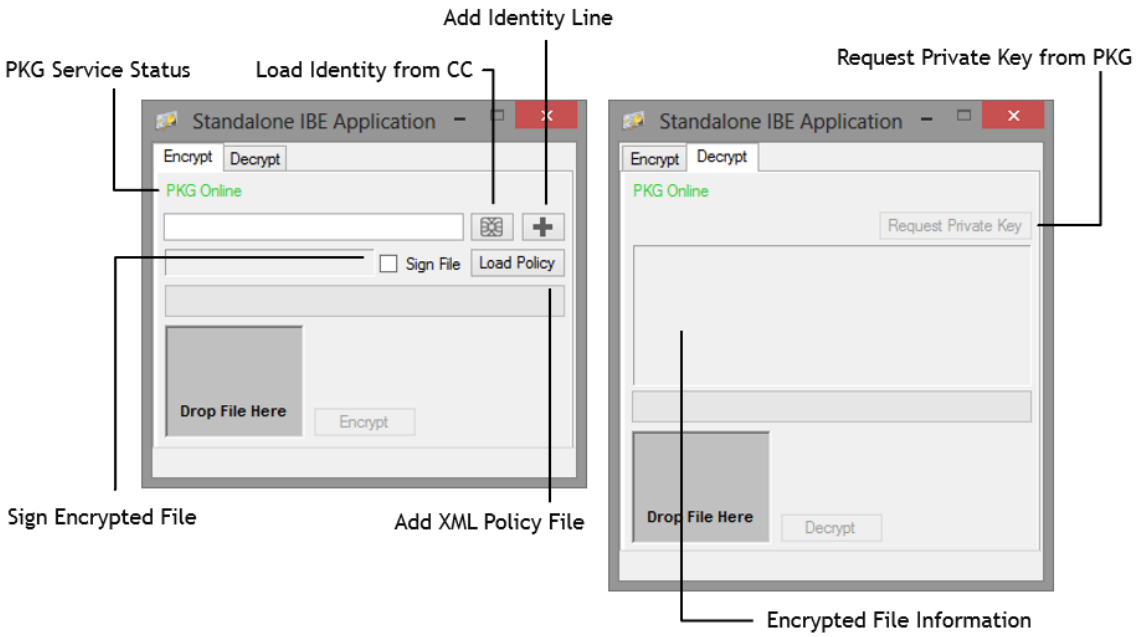


Figura 5.2: Visão geral da aplicação para *Windows Desktop*

5.3.1 Funcionamento Geral

Pretende-se com esta aplicação fornecer uma interface simples e intuitiva para o utilizador na cifra e decifra de ficheiros, e assim fornecer uma prova de conceito da arquitetura desenvolvida. A **Figura 5.2** fornece uma visão geral da aplicação e das suas funcionalidades, estas serão apresentadas com mais detalhe ao longo deste capítulo.

5.3.2 Ligação ao PKG e atualização de parâmetros

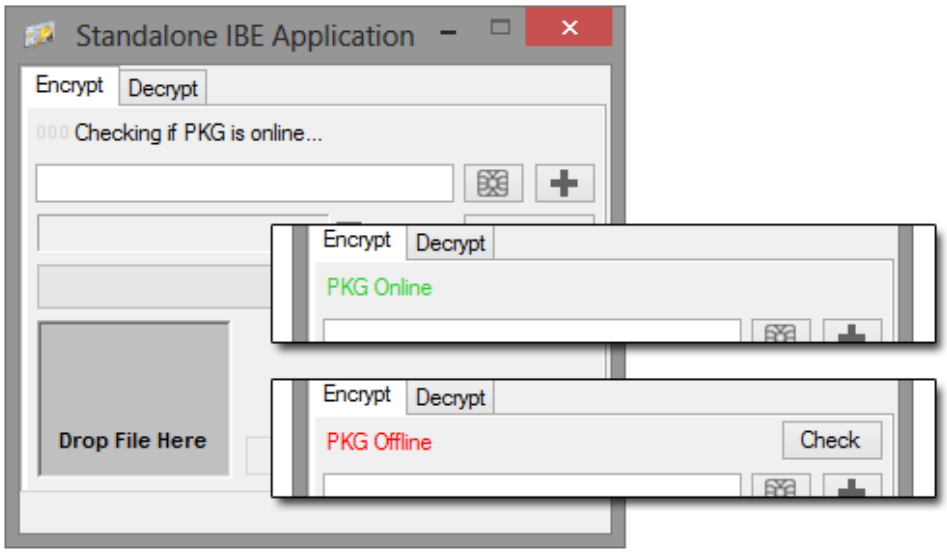


Figura 5.3: Informação sobre a conetividade ao serviço de PKG

Como foi referido no capítulo anterior, assim que a aplicação é iniciada, esta vai testar primeiramente a conetividade ao serviço PKG. O endereço do PKG pode ser definido pelo utilizador a

qualquer momento num ficheiro de configuração que é parte da aplicação. Se conseguir estabelecer ligação e iniciar uma sessão SSL a aplicação irá primeiro verificar se possui os parâmetros e a definição de curva elíptica, caso não os possua, vai pedir estes elementos ao PKG, caso contrário vai efetuar o *hash* dos elementos e verificar se estes coincidem com os elementos presentes no PKG e efetuar a respetiva atualização se necessário.

Se a aplicação não conseguir estabelecer ligação (ver **Figura 5.3**), o utilizador pode verificar a qualquer altura a conetividade, utilizando para tal o botão "Check". Sem ligação estabelecida, o utilizador pode apenas cifrar ficheiros, a assinatura e a decifra não estão disponíveis, pois é necessário obter uma chave privada junto do PKG.

5.3.3 Cifra de ficheiros

Na cifra de ficheiros, o utilizador começa por introduzir as identidades, até um máximo de cinco. Este limite é imposto pela aplicação e não pela arquitetura proposta, assim é perfeitamente possível a criação de aplicações com a biblioteca ou com os *wrappers* apresentados, que permitam a cifra para *n* identidades. O utilizador pode introduzir estas identidades manualmente, ou carregar a identidade a partir do Cartão de Cidadão, utilizando o botão representado na imagem por um *smartcard*.

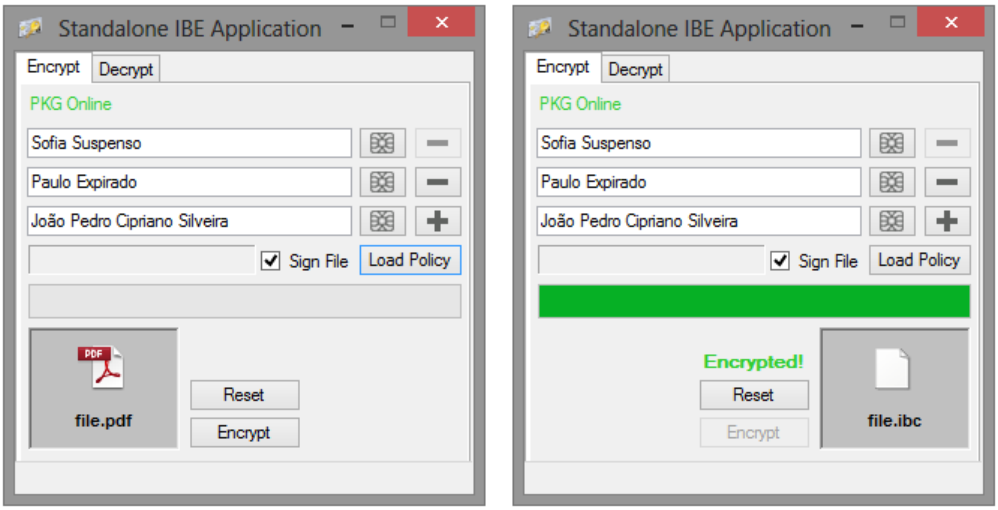


Figura 5.4: Cifra de um ficheiro com múltiplas identidades

Para o processo de cifra, o utilizador pode ainda selecionar se deseja ou não assinar o ficheiro cifrado com a sua identidade (através da opção "Sign File"), de modo a realizar este processo, é necessário que o utilizador se autentique ao PKG. Para além desta opção, a aplicação permite ainda carregar e associar um ficheiro XML com uma definição das políticas à(s) chave(s) pública(s) com a qual/quais o ficheiro será cifrado através do botão "Load Policy". Caso sejam associadas políticas, estas são guardadas juntamente com o ficheiro cifrado como mostrado anteriormente.

O utilizador pode selecionar o ficheiro que deseja cifrar, simplesmente arrastando-o para a área assinalada na aplicação. Depois de cifrado, pode arrastar o ficheiro resultante para a pasta/local pretendida/o. Na **Figura 5.4** o ficheiro "file.pdf"é cifrado com três identidades, "Sofia Suspenso", "Paulo Expirado"e "João Pedro Cipriano Silveira", e assinado com a identidade do utilizador, o ficheiro cifrado resultante é "file.ibc".

5.3.4 Decifra de ficheiro

A janela de decifra na aplicação, permite, ao arrastar um ficheiro cifrado para o campo destinado, mostrar informação sobre este ao utilizador (ver **Figura 5.5**). Esta informação consiste no nome original e tamanho do ficheiro, identidades para as quais foi cifrado e caso tenha sido assinado e a assinatura respetiva seja verificada com sucesso pela aplicação, é mostrado por quem foi assinado o ficheiro, no caso de a assinatura não ser verificada com sucesso, a aplicação irá mostrar a mensagem "Unable to verify signature"ou caso não tenha sido assinado "File not signed". Contudo a decisão de decifrar ou não o ficheiro cabe sempre ao utilizador da aplicação.

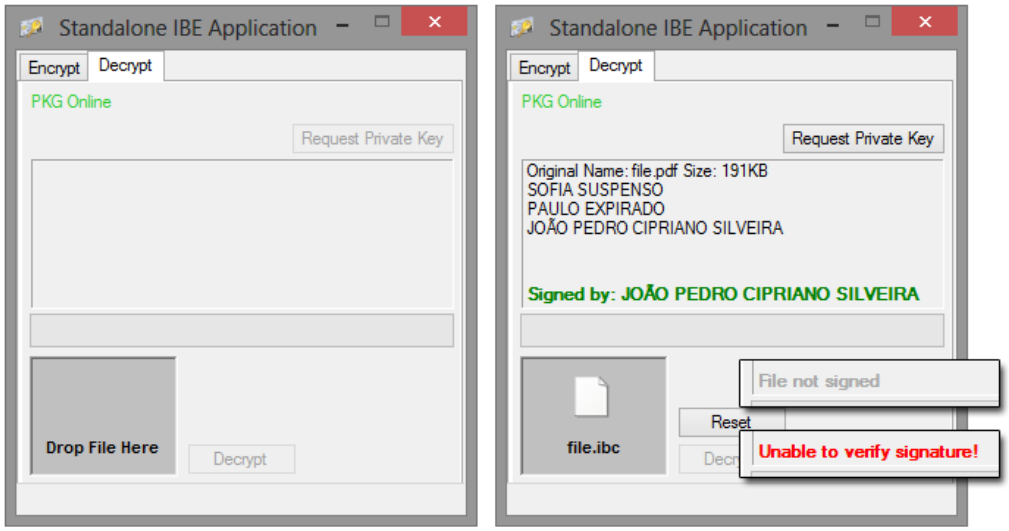


Figura 5.5: Informação sobre o ficheiro cifrado

Para decifrar o ficheiro, o utilizador tem primeiro que efetuar um pedido da chave privada necessária ao PKG, utilizando para tal o botão "Request Private Key". Este botão vai desencadear o processo de autenticação do utilizador junto do PKG. Após a obtenção da chave com sucesso, o utilizador pode então decifrar e obter o ficheiro original (**Figura 5.6**).

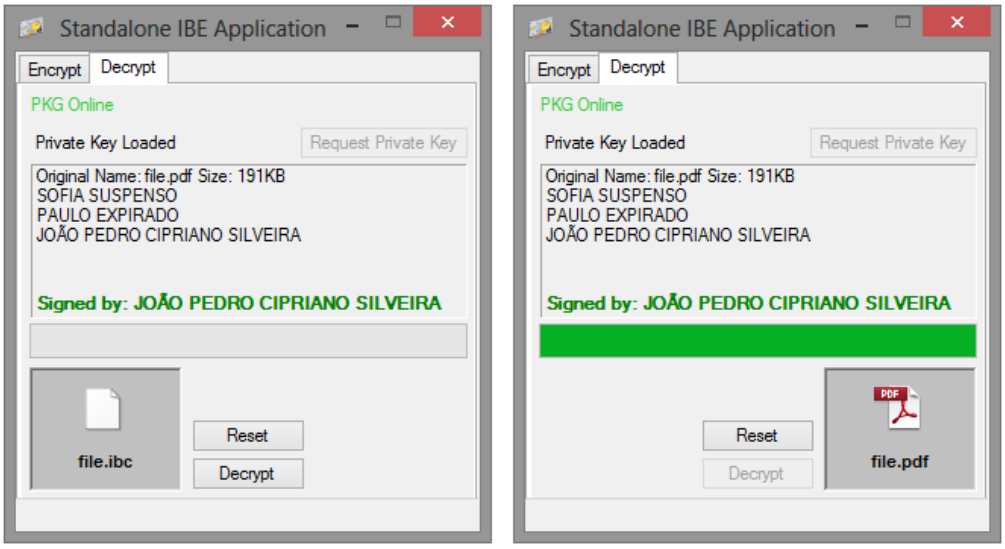


Figura 5.6: Obtenção de chave privada e decifra do ficheiro

5.3.5 Autenticação

De modo a obter a chave privada, para o processo de assinatura do ficheiro cifrado e para a decifra, é necessário que o utilizador se autentique com o seu CC através da aplicação, junto do PKG. Para tal, é necessário que utilizador tenha ligação à Internet e que o serviço de PKG esteja disponível.

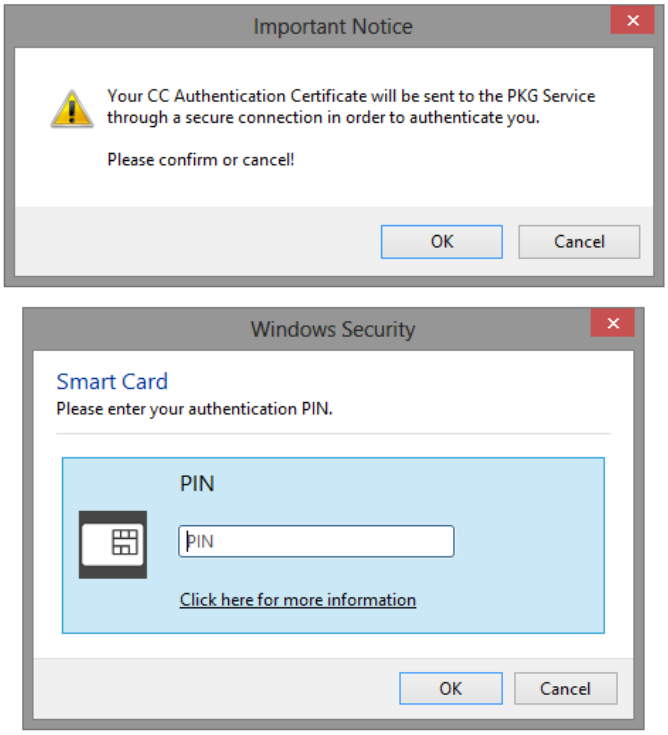


Figura 5.7: Aviso apresentado durante o processo de autenticação e pedido de introdução do PIN de autenticação do CC

Todo o processo é efetuado em alguns segundos. O utilizador é informado que o certificado de autenticação do CC será enviado para o PKG e o processo só continua após a confirmação desta aceitação. Assim que a aplicação recebe o *GUID* para assinar (ver **Secção 4.3.6**), é pedido ao utilizador que introduza o PIN de autenticação do CC, de forma a poder utilizar a chave privada correspondente. Na **Figura 5.7** podemos observar o aviso apresentado para aceitação pelo utilizador e a janela para a introdução do PIN exibido via *Windows Cryptographic Service Provider* (Windows CSP). Numa arquitetura não *windows* poderia ser utilizado PKCS#11 para o mesmo efeito uma vez que as duas interfaces são suportadas pelo CC.

5.4 Serviço PKG no *Azure*

De forma a garantir a disponibilidade do PKG, este foi alojado como serviço WCF (Windows Communication Foundation) no *Windows Azure*. Este serviço é utilizado pela aplicação *Windows Desktop* e pode ser facilmente utilizado por outras aplicações desenvolvidas em diferentes linguagens e sobre várias plataformas e o facto de estar alojado na *Cloud* ajuda a garantir a escalabilidade do sistema.

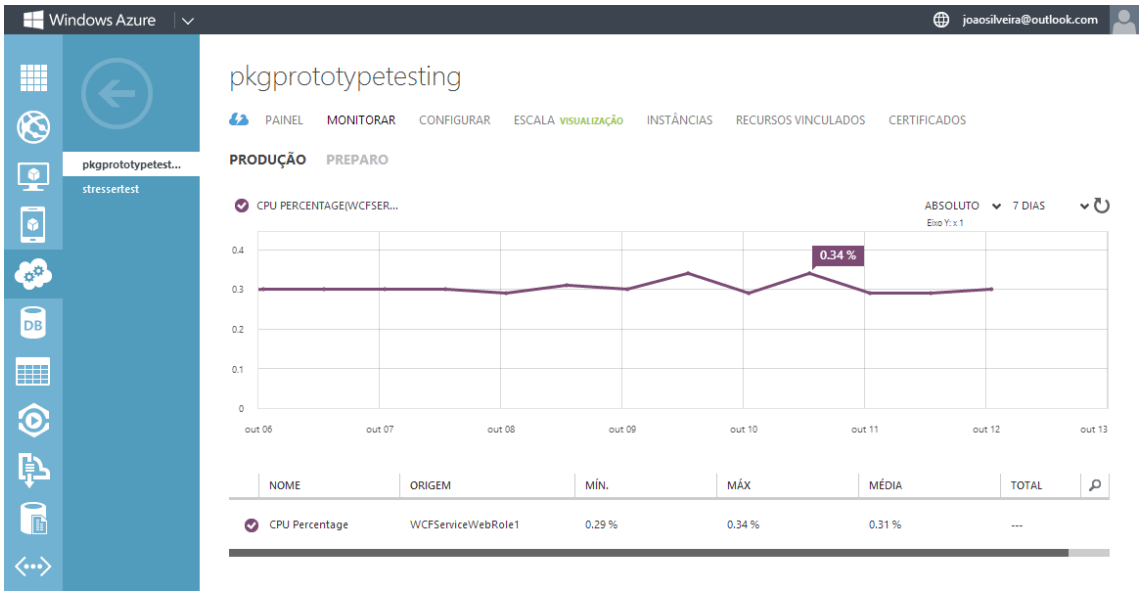


Figura 5.8: Painel de controlo do serviço PKG na plataforma *Windows Azure*

O serviço está a usar um único CPU (apesar deste número pode ser aumentado a qualquer momento numa questão de segundos) e foi configurado para suportar um máximo de 200 sessões ou 200 pedidos em acesso concorrente. Durante os testes efetuados observamos que o processo de autenticação de um utilizador e a respetiva geração de chaves, têm um custo computacionalmente baixo. Nos testes efetuados o valor máximo de utilização atingido pelo CPU foi de 0.34%, mesmo com vários utilizadores em acesso concorrente (**Figura 5.8**).

Foram também efetuados alguns testes de *stress*, com dois computadores diferentes em simultâneo a executar 25 *threads* cada um, em que cada *thread* é correspondente a uma sessão em que o utilizador contacta o servidor, autentica-se com o seu certificado e recebe a chave privada correspondente. Após vários teste, obtivemos um resultado de em média 19 segundos para a realização de 50 processos de autenticação e obtenção de chaves (cada processo corresponde ao upload do certificado para o serviço, verificação, autenticação, geração e obtenção de chave privada, logo o tempo obtido é dependente de vários fatores para além do PKG como os computadores de origem e a velocidade e qualidade da ligação à internet).

A partir dos resultados obtidos acreditamos que o serviço é capaz de dar resposta a um número elevado de pedidos apenas com um CPU, este número de CPUs é virtual e pode ser a qualquer momento aumentado para responder às necessidades dos utilizadores do serviço.

5.5 Aplicação Web para gestão e cifra de ficheiros na *Cloud*

A biblioteca desenvolvida e apresentada na **Secção 3.5** é utilizada por uma aplicação Web, criada no âmbito de uma dissertação desenvolvida paralelamente no Projeto *PRICE*.

Esta aplicação fornece acesso web uniforme aos fornecedores de armazenamento e serviços na *Cloud* mais populares, tais como a *Dropbox*, *Skydrive*, *Meo Cloud* e *Google Drive* (ver **Figura 5.9**), e usa o mecanismo de autenticação do CC como *Token* de acesso único, contudo funciona de forma diferente da aplicação anteriormente apresentada.

O PKG está integrado na aplicação Web, assim, tanto a cifra e decifra de ficheiros como o cumprimento das políticas é da responsabilidade da própria aplicação. Esta forma de implementação é especialmente interessante na definição e cumprimento de políticas e regras baseadas em papeis, uma vez que a própria aplicação é responsável por estes dois processos e contêm informação sobre o utilizador.

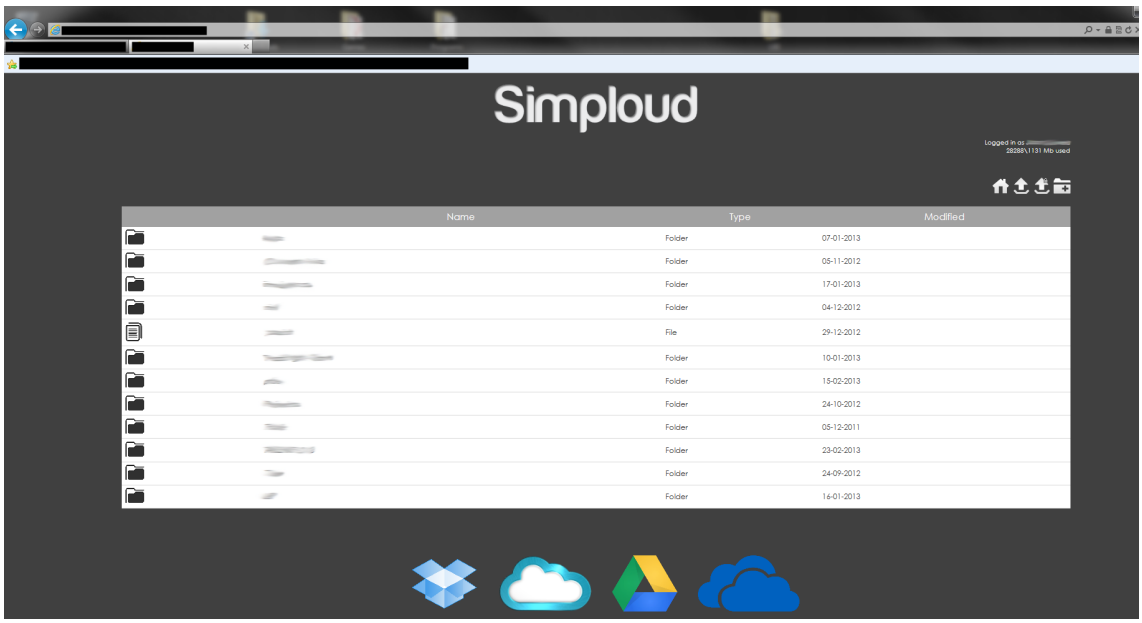


Figura 5.9: Aplicação Web para a gestão e cifra de ficheiros na *Cloud*

5.6 Tempos de execução

Na **Tabela 5.1** é possível observar os tempos de cifra do ficheiro e de chave obtidos na aplicação *Windows Desktop*, para vários ficheiros gerados aleatoriamente com tamanhos diferentes. O tempo de cifra da chave permanece constante como seria de esperar, uma vez que este processo cifra apenas a chave e é independente do tamanho do ficheiro. É importante relembrar que é utilizado um sistema de criptografia híbrida, em que o ficheiro em si é cifrado com o algoritmo AES em modo *Cipher Block Chaining* (CBC), com chave de 256 bits e esta chave é cifrada com IBE, nomeadamente utilizando o esquema *Emribe* apresentado no **Capítulo 3** com uma curva elíptica de tamanho/ordem de 256 bits.

File Size	Key Encryption Time (seconds)	File Encryption Time (seconds)
100 KB	0.0792	0.0056
1 MB	0.0797	0.0388
10 MB	0.0802	0.3638
100 MB	0.0782	3.5612
500 MB	0.0789	17.8458
1 GB	0.0780	37.8323

Tabela 5.1: Tempos de execução de cifra para ficheiros com diferentes tamanhos

A observação feita para a **Tabela 5.1**, aplica-se igualmente para os resultados obtidos na decifra de ficheiros e apresentados na **Tabela 5.2**. Os tempos de decifra para os mesmos ficheiros, são similares aos observados na cifra, enquanto que os tempos de decifra da chave são inferiores, uma vez que são necessárias menos operações sobre a curva elíptica neste processo, em comparação com a cifra da chave.

File Size	Key Decryption Time (seconds)	File Decryption Time (seconds)
100 KB	0.0193	0.0063
1 MB	0.0194	0.0421
10 MB	0.0188	0.3815
100 MB	0.0198	3.8550
500 MB	0.0195	19.3742
1 GB	0.0196	39.2573

Tabela 5.2: Tempos de execução de decifra para ficheiros com diferentes tamanhos

A **Tabela 5.3** contém os tempos de cifra da chave para múltiplas identidades. Observamos que a adição de uma identidade na cifra da chave têm um custo bastante reduzido de aproximadamente 20 milissegundos. Para a adição de 15 identidades e cifra de um ficheiro de 10 MB teríamos um tempo de cifra de chave de aproximadamente 360 milissegundos e 380 milissegundos para algoritmo AES, resultando num tempo global de cifra de aproximadamente 740 milissegundos. Contudo este tempo global pode ser reduzido.

n-identities	Key Encryption Time (seconds)
1	0.0789
2	0.1007
3	0.1206
4	0.1411
5	0.1618

Tabela 5.3: Tempos de execução na cifra da chave para n-identidades

Para o caso da adição de mais identidades sugerimos a paralelização do processo de cifra de chaves como forma de diminuir o tempo global de cifra. Se olharmos para o processo de cifra do esquema *Emribe* apresentado na **Secção 3.2.2** é possível verificar facilmente que o cálculo do elemento $\boldsymbol{V}$, necessário para cada identidade, pode ser paralelizado. Assim, para 5 identidades o resultado de cifra da chave será:

$$C = (U, V_1, V_2, V_3, V_4, V_5, W1, W2, \mathcal{L}, \sigma), \text{ onde } V_i = r \cdot F_{ID_i} + r \cdot Q$$

Cada V_i corresponderá a uma *thread* T_i, assim para este caso teremos 5 *threads*. Como é possível observar na **Tabela 5.4**, esta paralelização resultou num decréscimo de mais de 50% do tempo de execução de cifra de chave. No caso de 5 identidades passámos de 161 para 93 milissegundos, ficando este valor abaixo do resultado obtido para 2 identidades numa cifra de chave em série. É também interessante observar que o tempo de cifra de chave com 10 identidades em modo série

e 25 identidades em modo paralelo (com threads) é semelhante, o que evidencia o aumento substancial de eficácia no tempo de cifra. Estes testes foram efetuados num computador com quatro *cores* físicos, e os tempos são referentes ao tempo total da cifra de chave (cálculo dos elementos *U*, *W1*,*W2*,$\mathcal{L}$ e σ em série, e cálculo dos V_i em paralelo no caso da utilização de *threads*).

n-identities	Key Encryption Time Normal (seconds)	Key Encryption Time /w Threads (seconds)
5	0.1618	0.0935
10	0.2527	0.1265
15	0.3563	0.1770
20	0.4564	0.2135
25	0.5219	0.2475

Tabela 5.4: Comparação dos tempos de execução na cifra da chave para n-identidades, em modo série e em paralelo

Para o caso da assinatura de um ficheiro, registou-se um tempo médio de 50 milissegundos. No caso da autenticação e geração de chave, o tempo médio da operação foi cerca de 18 segundos. Este tempo é condicionado por fatores como a velocidade e estabilidade da ligação à Internet e as operações sobre o *smartcard* do CC (que são bastante lentas), neste caso a necessidade de introdução do PIN de autenticação para a utilização da chave privada respetiva, tem especial influência no tempo médio obtido.
Na adição de políticas na cifra de um ficheiro, esta operação não vai influenciar o tempo de cifra da chave, nem o tempo de cifra do ficheiro. A única operação extra que necessita de ser efetuada, é o *hash* das políticas e a sua adição na chave pública, operações que podemos considerar como irrelevantes em termos computacionais.

Todos os testes à aplicação, aqui representados, foram efetuados numa máquina com processador *Intel i7-3610QM, 2.30Ghz*. Os resultados apresentados correspondem à média dos valores obtidos, após várias repetições do mesmo teste.

5.7 Conclusão

Neste capítulo foram apresentados os resultados obtidos, nomeadamente a aplicação desenvolvida para *Windows* de cifra e decifra de ficheiros e que corresponde a um cliente na arquitetura proposta e a solução para fornecimento de acesso web uniforme a fornecedores de armazenamento e serviços em *Cloud*, que aplica a biblioteca criada, especificada no **Capítulo 3**.
Foi desenvolvida uma aplicação para facilitar a criação de políticas XML, esta permite ao utilizador escolher de forma gráfica as regras/especificações e gerar automaticamente um ficheiro XML com as opções selecionadas.

Em relação à aplicação para *Windows Desktop* foi apresentado o funcionamento geral da ferramenta. Especificamente foi apresentada cada uma das suas funcionalidades, como a ligação ao PKG e atualização de parâmetros, a cifra de ficheiros com as opções de cifrar com um máximo de cinco identidades, adicionar assinatura e carregar um ficheiro de políticas XML e a respetiva

decifra de ficheiros. Foi igualmente apresentado o processo de autenticação na aplicação, onde é necessário utilizar o PIN de autenticação do CC de forma a poder usar a sua chave privada de autenticação.
Os tempos de execução obtidos nesta aplicação foram bastante satisfatórios na assinatura e na cifra de chaves, tanto para uma, como para várias identidades, bem como para os tempos do processo de decifra. Para a autenticação obtivemos um resultado pior que o esperado, facto que se justifica pelas velocidades bastante lentas das operações sobre o CC.

É feita uma breve descrição da aplicação web para o acesso uniforme aos vários fornecedores de armazenamento e serviços na *Cloud*, desenvolvida de forma paralela no âmbito de outra dissertação de mestrado dentro do Projeto *PRICE*.

Foi igualmente apresentado o serviço de PKG desenvolvido e que se encontra a correr online no *Azure*, que suporta acesso concorrente e que consideramos a partir dos resultados obtidos como computacionalmente bastante eficiente.

O objetivo deste capítulo consiste na apresentação de um protótipo final da aplicação cliente e do serviço PKG. Ambos desenvolvidos como teste à arquitetura proposta no capítulo anterior.

Capítulo 6

Conclusão e trabalho futuro

O tema desta dissertação surgiu da parceria entre o laboratório RELEASE da Universidade da Beira Interior e a *PT Inovação* de Aveiro, no âmbito do projeto PRICE, cujo o objetivo é o desenvolvimento e aplicação de tecnologias para a privacidade em ambientes de computação em *Cloud*, e em particular o uso de criptografia baseada em identidade em armazenamento na *Cloud* e políticas de privacidade.

No inicio do trabalho foram identificados e propostos os objetivos a atingir na realização desta dissertação. A investigação e o trabalho levado a cabo durante o decorrer da dissertação permitiu que estes objetivos fossem alcançados com sucesso.

6.1 Conclusões

No primeiro capítulo, foi feita uma introdução à temática da dissertação, apresentada as motivações, indicados os objetivos principais propostos para este trabalho, forma de abordagem à investigação e apresentada a estrutura desta dissertação.

No segundo capítulo, fez-se um levantamento do estado da arte e das tecnologias utilizadas no decorrer do trabalho. Foi apresentado o Cartão de Cidadão e as suas características como documento de identificação e *smartcard* com especial ênfase na segurança, os fundamentos de criptografia de curva elíptica e o aparecimento da criptografia baseada em identidade (assinatura e cifra). Foram expostas as vantagens da utilização de um sistema IBC, os problemas em aberto, bem como as soluções existentes nesta área.

No capítulo três foram apresentadas as descrições técnicas dos esquemas implementados e as implementações, que levaram à criação de um conjunto de *wrappers* para a linguagem *.NET C#* e ao desenvolvimento de uma biblioteca para implementação de um sistema completo de IBE que tem por base o esquema *Emribe*.

No capítulo seguinte foi apresentada a solução proposta para a arquitetura de cifra e decifra de ficheiros com IBE e com o CC como mecanismo de autenticação junto do centro gerador de chaves (PKG). Foram abordadas a estrutura geral do sistema e os diferentes processos que a compõem, bem como as respetivas justificações para as decisões efetuadas.

Por fim, no último capítulo foram expostos os resultados obtidos, nomeadamente a aplicação protótipo desenvolvida, a integração do sistema numa plataforma Web já existente e o serviço PKG, que é o elemento central na arquitetura. Foram apresentados também vários tempos de execução obtidos durante as várias fases de funcionamento da arquitetura.

Os objetivos a alcançar, definidos no início da dissertação, foram cumpridos com sucesso e resultaram nas seguintes contribuições:

- Estudo sobre os sistemas criptográficos baseados em identidade, os esquemas subjacentes e as soluções existentes;
- Implementação de um esquema de assinatura e de dois esquemas de cifra e comparação a nível de custo computacional;
- Criação de vários *wrappers* para a utilização dos esquemas referidos na linguagem *.NET C#* e de uma biblioteca para simplificar a criação de sistemas IBC, em que a ideia é fornecer uma espécie de *plugin* que possa ser utilizado por outras aplicações;
- Definição das características de identidade a utilizar e especificação de uma arquitetura baseada num centro de geração de chaves (PKG) e os respetivos de autenticação com o CC, geração de chaves e revogação;
- Implementação de uma aplicação de cifra e decifra de ficheiros e integração destas capacidades numa aplicação Web existente de fornecimento de acesso uniforme a fornecedores de armazenamento e serviços em *Cloud*, de forma a mostrar a viabilidade da solução proposta;
- Implementação de um sistema simples de políticas de privacidade de ficheiros e integração na arquitetura criada.

Na base da arquitetura criada, está um longo trabalho de investigação, que permitiu associar os resultados produzidos a um sistema, no qual acreditamos estar a par das necessidades de segurança do mercado. Em particular pequenas e médias empresas, organizações e serviços públicos ou privados que necessitam de assegurar a confidencialidade e integridade das informações transmitidas dentro dos seus serviços de informação.

No primeiro capítulo referimos como serviços básicos de segurança que um sistema criptográfico deve fornecer a **Confidencialidade**, **Integridade**, **Autenticação** e **Não-Repudiação**. Acreditamos que a arquitetura proposta e as soluções apresentadas nesta dissertação vão de encontro a estes requisitos.

Concluindo, os resultados obtidos foram bastante satisfatórios e confiamos que a implementação prática da nossa solução num contexto real, pode ser vantajosa, em relação aos sistemas de informação criptográficos tradicionais.

A nível pessoal, o trabalho desenvolvido durante o decorrer da dissertação foi bastante gratificante, permitiu o contacto com uma variedade de conceções e tecnologias que eram numa fase inicial desconhecidos e possibilitou uma aprendizagem importante a nível tecnológico nas áreas de *smartcards* e segurança, com ênfase especial na criptografia.

6.2 Trabalho futuro

Em termos de trabalho futuro e futuras direções de investigação, estes podem ser divididos em aspetos de implementação e novas funcionalidades.

6.2.1 Aspetos de Implementação

No que diz respeito aos aspetos de implementação propõe-se:

- Implementação de um *Public Parameter Server* (PPS), desta forma o PKG é substituído na disponibilização de parâmetros e definição de curva. Assim este irá ter como função exclusiva a geração de chaves privadas;
- Explorar e implementar mecanismos que permitam eliminar os problemas de custódia de chaves inerentes do esquema utilizado;
- Apesar de em certos aspetos a arquitetura apresentada nesta dissertação ser muito semelhante à descrita no *RFC5408* [AMS09], seria interessante explorar esta arquitetura e seguir o *RFC5409* [MS09] de forma a implementar a construção de mensagens cifradas utilizando CMS;
- Permitir múltiplas assinaturas, ou seja, possibilitar que um ficheiro cifrado seja assinado por um conjunto de utilizadores.

6.2.2 Novas Funcionalidades

Como novas funcionalidades propõe-se:

- A definição e adição de novas regras para estender as políticas de ficheiros existentes, como por exemplo a adição de uma política que permita definir a gama de IP a partir dos quais o utilizador pode obter a chave privada com sucesso;
- Possibilitar ao utilizador armazenar as chaves privadas obtidas do PKG para utilizar posteriormente ou delegar.

Bibliografia

[AG09] Giuseppe Ateniese and Paolo Gasti. Universally anonymous ibe based on the quadratic residuosity assumption. In *Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, CT-RSA '09, pages 32–47, Berlin, Heidelberg, 2009. Springer-Verlag. http://dx.doi.org/10.1007/978-3-642-00862-7_3. 15

[AGM+13] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013. http://dx.doi.org/10.1007/s13389-013-0057-3. 20

[AHK+03] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). Technical report, IBM, 2003. http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html. 38

[AMA12] Agência para a Modernização Admnistrativa AMA. Manual técnico do middleware cartão de cidadão, 2012. 10

[AMS09] G. Appenzeller, L. Martin, and M. Schertler. Identity-Based Encryption Architecture and Supporting Data Structures. RFC 5408 (Informational), January 2009. http://www.ietf.org/rfc/rfc5408.txt. 15, 59

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Christian Cachin and JanL. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin Heidelberg, 2004. http://dx.doi.org/10.1007/978-3-540-24676-3_14. 15, 18

[BBB+05] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management - part 1: General. In *NIST Special Publication 800-57, August 2005, National Institute of Standards and Technology. Available at http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf*, 2005. 12

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag. http://dl.acm.org/citation.cfm?id=646766.704155. 14, 17, 18, 21

[BGH07] Dan Boneh, Craig Gentry, and Michael Hamburg. Space-efficient identity based encryption without pairings. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:647–657, 2007. 14

[BM07] X. Boyen and L. Martin. Identity-Based Cryptography Standard (IBCS) #1: Supersingular Curve Implementations of the BF and BB1 Cryptosystems. RFC 5091 (Informational), December 2007. http://www.ietf.org/rfc/rfc5091.txt. 15

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer*

and communications security, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM. http://doi.acm.org/10.1145/168588.168596. 14

[BSNS05] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Efficient multi-receiver identity-based encryption and its application to broadcast encryption. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 380–397. Springer Berlin Heidelberg, 2005. http://dx.doi.org/10.1007/978-3-540-30580-4_26. 21

[BZ04] Joonsang Baek and Yuliang Zheng. Identity-based threshold decryption. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *Public Key Cryptography - PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 262–276. Springer Berlin Heidelberg, 2004. http://dx.doi.org/10.1007/978-3-540-24632-9_19. 17

[Cer] CertiVox. Miracl cryptographic sdk. Disponível a 28 de Agosto de 2013. http://www.certivox.com/miracl/. 20

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *Proceedings of Eurocrypt 2003*, pages 255–271. Springer-Verlag, 2003. 15

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer Berlin Heidelberg, 2001. http://dx.doi.org/10.1007/3-540-45325-3_32. 14, 16

[CZKK05] Xiaofeng Chen, Fangguo Zhang, DivyanM. Konidala, and Kwangjo Kim. New id-based threshold signature scheme from bilinear pairings. In Anne Canteaut and Kapaleeswaran Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 371–383. Springer Berlin Heidelberg, 2005. http://dx.doi.org/10.1007/978-3-540-30556-9_29. 17

[DCI11] A. De Caro and V. Iovino. jpbc: Java pairing based cryptography. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 850–855, 2011. 20

[DQ87] Yvo Desmedt and Jean-Jacques Quisquater. Public-key systems based on the difficulty of tampering. In AndrewM. Odlyzko, editor, *Advances in Cryptology - CRYPTO' 86*, volume 263 of *Lecture Notes in Computer Science*, pages 111–117. Springer Berlin Heidelberg, 1987. http://dx.doi.org/10.1007/3-540-47721-7_9. 14

[For] Inc. Fortinet. Email security solution appliances for anti-malware, anti spam, antivirus | fortinet. Disponível a 28 de Agosto de 2013. http:///www.fortinet.com/products/fortimail/index.html. 19

[FR94] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62:865–874, 1994. 14

[Gem] Inc. Gemalto. Gemplus develops the world's first identity-based encryption for smart cards. Disponível a 28 de Agosto de 2013. http://www.gemalto.com/press/gemplus/2004/id_security/02-11-2004-Identity-Based_Encryption.htm. 19

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. *Secure Distributed Key Generation for Discrete-Log Based Cryptosystems*. 1999. 18

[GM03] Simon Godik and Tim Moses, editors. *eXtensible Access Control Markup Language (XACML) Version 1.0*. February 2003. 38

[GS02] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. Cryptology ePrint Archive, Report 2002/056, 2002. http://eprint.iacr.org/. 15

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In LarsR. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481. Springer Berlin Heidelberg, 2002. http://dx.doi.org/10.1007/3-540-46035-7_31. 15

[Jou04] Antoine Joux. A one round protocol for tripartite diffie?hellman. *Journal of Cryptology*, 17(4):263–276, 2004. http://dx.doi.org/10.1007/s00145-004-0312-y. 14

[KG09] Aniket Kate and Ian Goldberg. Asynchronous distributed private-key generators for identity-based cryptography. Cryptology ePrint Archive, Report 2009/355, 2009. http://eprint.iacr.org/. 18

[Kob87] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48, 1987. 11

[KSW02] Günter Karjoth, Matthias Schunter, and Michael Waidner. Platform for enterprise privacy practices: Privacy-enabled management of customer data. pages 69–84. Springer, 2002. 38

[Lyn02] Ben Lynn. Authenticated identity-based encryption. Cryptology ePrint Archive, Report 2002/072, 2002. http://eprint.iacr.org/. 15

[Lyn08] Ben Lynn. Pbc library - the pairing-based cryptography library. http://crypto.stanford.edu/pbc, 2008. Disponível a 28 de Agosto de 2013. 19, 24

[MBDH] Marco Casassa Mont, Pete Bramhall, Chris R. Dalton, and Keith Harrison. A flexible role-based secure messaging service: Exploiting ibe technology in a health care trial. Technical report, Trusted Systems Laboratory, HP Laboratories Bristol. 19

[Mil86a] Victor S. Miller. Short programs for functions on curves. In *IBM Thomas J. Watson Research Center*, 1986. 14

[Mil86b] Victor S Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc. http://dl.acm.org/citation.cfm?id=18262.25413. 11

[MPB03] Marco Casassa Mont, Siani Pearson, and Pete Bramhall. Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services. pages 377–382. IEEE Computer Society, 2003. 38

[MS09] L. Martin and M. Schertler. Using the Boneh-Franklin and Boneh-Boyen Identity-Based Encryption Algorithms with the Cryptographic Message Syntax (CMS). RFC 5409 (Informational), January 2009. http://www.ietf.org/rfc/rfc5409.txt. 15, 59

[MUL11a] S.A. MULTICERT. Declaração de práticas de certificação da ec de assinatura digital qualificada do cartão de cidadão. Políticas, Ministério da Justiça, 2011. 9

[MUL11b] S.A. MULTICERT. Declaração de práticas de certificação da ec de autenticação do cartão de cidadão. Políticas, Ministério da Justiça, 2011. 9

[MVO91] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 80–89, New York, NY, USA, 1991. ACM. http://doi.acm.org/10.1145/103418.103434. 14

[MY91] UeliM. Maurer and Yacov Yacobi. Non-interactive public-key cryptography. In DonaldW. Davies, editor, *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 498–507. Springer Berlin Heidelberg, 1991. http://dx.doi.org/10.1007/3-540-46416-6_43. 14

[NSA05] NSA. Nsa suite B cryptography. Last accessed 14-Jul-2010. http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml, 2005. http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml. 11

[Pat02] Kenneth G. Paterson. ID-based signatures from pairingson elliptic curves. *Electronics Letters*, 38, 2002. 21

[RSK01] K. Ohgishi R. Sakai and M. Kasahara. Cryptosystems based on pairing over elliptic curve. The 2001 Symposium on Cryptography and Information Security, 2001. 14

[SCW06] Jun Shao, Zhenfu Cao, and Licheng Wang. Efficient id-based threshold signature schemes without pairings. Cryptology ePrint Archive, 2006, 2006. http://eprint.iacr.org/2006/308.pdf. 18

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. http://doi.acm.org/10.1145/359168.359176. 17

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc. http://dl.acm.org/citation.cfm?id=19478.19483. 2, 12, 14

[SK03] Ryuichi Sakai and Masao Kasahara. Id based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054, 2003. 18, 19

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology ? EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer Berlin Heidelberg, 2005. http://dx.doi.org/10.1007/11426639_27. 15

[Tan88] Hatsukazu Tanaka. A realization scheme for the identity-based cryptosystem. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 340–349. Springer Berlin Heidelberg, 1988. http://dx.doi.org/10.1007/3-540-48184-2_29. 14

[TI89] S. Tsujii and T. Itoh. An id-based cryptosystem based on the discrete logarithm problem. *Selected Areas in Communications, IEEE Journal on*, 7(4):467–473, 1989. 14

[TM] Inc. Trend Micro. Email encryption solutions - secure email - trend micro usa. Disponível a 28 de Agosto de 2013. http://www.trendmicro.com/us/enterprise/network-web-messaging-security/email-encryption/index.html. 19

[VSa] Inc. Voltage Security. Email encryption securemail, email data protection , secure messaging and key management solutions | voltage security. Disponível a 28 de Agosto de 2013. http://www.voltage.com/products/securemail/. 18

[VSb] Inc. Voltage Security. Voltage security celebrates 10 years of identity-based encryption; rapidly growing commercial adoption within global 2000 companies | voltage security. Disponível a 28 de Agosto de 2013. http://voltage.com/pressreleases/PR110824-VoltageSecurity-celebrates-10years-of-IBE.htm. 18

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer Berlin Heidelberg, 2005. http://dx.doi.org/10.1007/11426639_7. 15

[WLD+02] Xin Wang, Guillermo Lao, Thomas DeMartini, Hari Reddy, Mai Nguyen, and Edgar Valenzuela. Xrml - extensible rights markup language. In Michiharu Kudo, editor, *XML Security*, pages 71–79. ACM, 2002. 38

[Zuq10] André Zuquete. *Segurança em Redes Informáticas*. FCA - Editora de Informática, Lisboa, 3rd edition, 2010. 7

Apêndice A

Anexos

A.1 *Wrapper C#* - Esquema *BF Fullident*

A.1.1 pkg.h

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<malloc.h>
#include<pbc/pbc.h>

__declspec(dllexport) unsigned char * __cdecl generateNewMasterKey(char *curve, int *size);
__declspec(dllexport) unsigned char * __cdecl generateNewPparam(char *curve, int *size);
__declspec(dllexport) unsigned char * __cdecl getPrivateKey(char *curve, unsigned char *Qidhash, ↩
    int len, unsigned char *sbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl getPpub(char *curve, unsigned char *Pbyte, unsigned ↩
    char *sbyte, int *size);
```

A.1.2 pkg.h

```
#include"pkg.h"

__declspec(dllexport) unsigned char * __cdecl generateNewMasterKey(char *curve, int *size)
{
   pairing_t pairing;
   element_t s;

   pairing_init_set_str(pairing, curve);
   element_init_Zr(s, pairing);
   element_random(s);
   *size = element_length_in_bytes(s);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, s);

   element_clear(s);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl generateNewPparam(char *curve, int *size)
{
   pairing_t pairing;
   element_t P;

   pairing_init_set_str(pairing, curve);
   element_init_G1(P, pairing);
   element_random(P);
   *size = element_length_in_bytes(P);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, P);

   element_clear(P);
   pairing_clear(pairing);

   return data;
}
```

```
__declspec(dllexport) unsigned char * __cdecl getPrivateKey(char *curve, unsigned char *Qidhash, ↩
    int len, unsigned char *sbyte, int *size)
{
   pairing_t pairing;
   element_t Qid, s, Did;

   pairing_init_set_str(pairing, curve);
   element_init_G1(Qid, pairing);
   element_init_G1(Did, pairing);
   element_init_Zr(s, pairing);
   element_from_hash(Qid, Qidhash, len);
   element_from_bytes(s, sbyte);
   element_mul_zn(Did, Qid, s);
   *size = element_length_in_bytes(Did);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, Did);

   element_clear(Qid);
   element_clear(Did);
   element_clear(s);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl getPpub(char *curve, unsigned char *Pbyte, unsigned ↩
    char *sbyte, int *size)
{
   pairing_t pairing;
   element_t P, s, Ppub;

   pairing_init_set_str(pairing, curve);
   element_init_G1(P, pairing);
   element_init_G1(Ppub, pairing);
   element_init_Zr(s, pairing);

   element_from_bytes(P, Pbyte);
   element_from_bytes(s, sbyte);

   element_mul_zn(Ppub, P, s);

   *size = element_length_in_bytes(Ppub);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, Ppub);

   element_clear(P);
   element_clear(Ppub);
   element_clear(s);
   pairing_clear(pairing);

   return data;
}
```

A.1.3 client.h

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
#include<pbc/pbc.h>

__declspec(dllexport) unsigned char * __cdecl encryptCalcH2(char *curve, unsigned char *rbyte, ↩
    unsigned char *Qidhash, int len, unsigned char *Ppubbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl encryptGetR(char *curve, unsigned char *rhash, int ↩
    len, int *size);
__declspec(dllexport) unsigned char * __cdecl encryptCalcU(char *curve, unsigned char *Pbyte, ↩
    unsigned char *rbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl decryptCalcH2(char *curve, unsigned char *Didbyte, ↩
    unsigned char *Ubyte, int *size);
```

A.1.4 client.c

```
#include"client.h"

__declspec(dllexport) unsigned char * __cdecl encryptCalcH2(char *curve, unsigned char *rbyte, ↩
    unsigned char *Qidhash, int len, unsigned char *Ppubbyte, int *size)
{
   pairing_t pairing;
   element_t r, Qid, gid, gidr, Ppub;

   pairing_init_set_str(pairing, curve);
   element_init_G1(Ppub, pairing);
   element_init_G1(Qid, pairing);
   element_init_Zr(r, pairing);
   element_init_GT(gid, pairing);
   element_init_GT(gidr, pairing);
   element_from_hash(Qid, Qidhash, len);
   element_from_bytes(r, rbyte);
   element_from_bytes(Ppub, Ppubbyte);
   element_pairing(gid, Qid, Ppub);
   element_pow_zn(gidr, gid, r);
   *size = element_length_in_bytes(gidr);
   unsigned char * data = malloc(*size);
   element_to_bytes(data, gidr);

   element_clear(Ppub);
   element_clear(Qid);
   element_clear(r);
   element_clear(gid);
   element_clear(gidr);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl encryptGetR(char *curve, unsigned char *rhash, int ↩
    len, int *size)
{
   pairing_t pairing;
   element_t r;

   pairing_init_set_str(pairing, curve);
   element_init_Zr(r, pairing);
   element_from_hash(r, rhash, len);
   *size = element_length_in_bytes(r);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, r);

   element_clear(r);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl encryptCalcU(char *curve, unsigned char *Pbyte, ↩
    unsigned char *rbyte, int *size)
{
   pairing_t pairing;
   element_t P, r, U;

   pairing_init_set_str(pairing, curve);
   element_init_G1(P, pairing);
   element_init_G1(U, pairing);
   element_init_Zr(r, pairing);
   element_from_bytes(r, rbyte);
   element_from_bytes(P, Pbyte);
   element_mul_zn(U, P, r);
   *size = element_length_in_bytes(U);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, U);
```

```
   element_clear(U);
   element_clear(P);
   element_clear(r);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl decryptCalcH2(char *curve, unsigned char *Didbyte, ↩
    unsigned char *Ubyte, int *size)
{
   pairing_t pairing;
   element_t Did, U, DidU;

   pairing_init_set_str(pairing, curve);
   element_init_G1(Did, pairing);
   element_init_G1(U, pairing);
   element_init_GT(DidU, pairing);
   element_from_bytes(Did, Didbyte);
   element_from_bytes(U, Ubyte);
   element_pairing(DidU, Did, U);
   *size = element_length_in_bytes(DidU);
   unsigned char * data = malloc(*size);
   element_to_bytes(data, DidU);

   element_clear(Did);
   element_clear(U);
   element_clear(DidU);
   pairing_clear(pairing);

   return data;
}
```

A.2 *Wrapper C#* - Esquema *Emribe*

A.2.1 pkg.h

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<malloc.h>
#include<pbc/pbc.h>

__declspec(dllexport) unsigned char * __cdecl generateNewMasterKey(char *curve, int *size);
__declspec(dllexport) unsigned char * __cdecl generateNewPparam(char *curve, int *size);
__declspec(dllexport) unsigned char * __cdecl generateNewQparam(char *curve, int *size);
__declspec(dllexport) unsigned char * __cdecl getPrivateKey(char *curve, unsigned char *H1hash, int↩
     H1len, unsigned char *sbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl calcPpub(char *curve, unsigned char *Pbyte, unsigned ↩
    char *sbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl calcpairQPpub(char *curve, unsigned char *Qbyte, ↩
    unsigned char *Ppubbyte, int *size);
```

A.2.2 pkg.c

```
#include"pkg.h"

__declspec(dllexport) unsigned char * __cdecl generateNewMasterKey(char *curve, int *size)
{
   pairing_t pairing;
   element_t s;

   pairing_init_set_str(pairing, curve);
   element_init_Zr(s, pairing);
   element_random(s);
   *size = element_length_in_bytes(s);
   unsigned char *data = malloc(*size);
```

```
    element_to_bytes(data, s);

    element_clear(s);
    pairing_clear(pairing);

    return data;
}

__declspec(dllexport) unsigned char * __cdecl generateNewPparam(char *curve, int *size)
{
    pairing_t pairing;
    element_t P;

    pairing_init_set_str(pairing, curve);
    element_init_G1(P, pairing);
    element_random(P);
    *size = element_length_in_bytes(P);
    unsigned char *data = malloc(*size);
    element_to_bytes(data, P);

    element_clear(P);
    pairing_clear(pairing);

    return data;
}

__declspec(dllexport) unsigned char * __cdecl generateNewQparam(char *curve, int *size)
{
    pairing_t pairing;
    element_t Q;

    pairing_init_set_str(pairing, curve);
    element_init_G1(Q, pairing);
    element_random(Q);
    *size = element_length_in_bytes(Q);
    unsigned char *data = malloc(*size);
    element_to_bytes(data, Q);

    element_clear(Q);
    pairing_clear(pairing);

    return data;
}

__declspec(dllexport) unsigned char * __cdecl getPrivateKey(char *curve, unsigned char *H1hash, int↩
      H1len, unsigned char *sbyte, int *size)
{
    pairing_t pairing;
    element_t H1, s, pk;

    pairing_init_set_str(pairing, curve);
    element_init_G1(H1, pairing);
    element_init_G1(pk, pairing);
    element_init_Zr(s, pairing);
    element_from_hash(H1, H1hash, H1len);
    element_from_bytes(s, sbyte);
    element_mul_zn(pk, H1, s);
    *size = element_length_in_bytes(pk);
    unsigned char *data = malloc(*size);
    element_to_bytes(data, pk);

    element_clear(H1);
    element_clear(pk);
    element_clear(s);
    pairing_clear(pairing);

    return data;
}
```

```
__declspec(dllexport) unsigned char * __cdecl calcPpub(char *curve, unsigned char *Pbyte, unsigned ↩
    char *sbyte, int *size)
{
   pairing_t pairing;
   element_t P, s, Ppub;

   pairing_init_set_str(pairing, curve);
   element_init_G1(P, pairing);
   element_init_G1(Ppub, pairing);
   element_init_Zr(s, pairing);
   element_from_bytes(P, Pbyte);
   element_from_bytes(s, sbyte);
   element_mul_zn(Ppub, P, s);
   *size = element_length_in_bytes(Ppub);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, Ppub);

   element_clear(P);
   element_clear(Ppub);
   element_clear(s);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl calcpairQPpub(char *curve, unsigned char *Qbyte, ↩
    unsigned char *Ppubbyte, int *size)
{
   pairing_t pairing;
   element_t Q, Ppub, eQPpub;

   pairing_init_set_str(pairing, curve);
   element_init_G1(Q, pairing);
   element_init_G1(Ppub, pairing);
   element_init_GT(eQPpub, pairing);
   element_from_bytes(Q, Qbyte);
   element_from_bytes(Ppub, Ppubbyte);
   element_pairing(eQPpub, Q, Ppub);
   *size = element_length_in_bytes(eQPpub);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, eQPpub);

   element_clear(Q);
   element_clear(Ppub);
   element_clear(eQPpub);
   pairing_clear(pairing);

   return data;
}
```

A.2.3 client.h

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
#include<pbc/pbc.h>

__declspec(dllexport) unsigned char * __cdecl genBigR(char *curve, int *size);
__declspec(dllexport) unsigned char * __cdecl genSmallR(char *curve, int *size);
__declspec(dllexport) unsigned char * __cdecl calcU(char *curve, unsigned char *Pbyte, unsigned ↩
    char *rbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl calcrQ(char *curve, unsigned char *Qbyte, unsigned ↩
    char *rbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl calcV(char *curve, unsigned char *rbyte, unsigned ↩
    char *H1hash, int H1len, unsigned char *rQbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl calcW1(char *curve, unsigned char *eQPpubbyte, ↩
    unsigned char *smallRbyte, unsigned char *bigRbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl decryptCalcR(char *curve, unsigned char *Ubyte, ↩
    unsigned char *pkbyte, unsigned char *Ppubbyte, unsigned char *Vbyte, unsigned char *W1byte, ↩
```

```
    int *size);
```

A.2.4 client.c

```
#include"client.h"

__declspec(dllexport) unsigned char * __cdecl genBigR(char *curve, int *size)
{
   pairing_t pairing;
   element_t R;

   pairing_init_set_str(pairing, curve);
   element_init_GT(R, pairing);
   element_random(R);
   *size = element_length_in_bytes(R);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, R);

   element_clear(R);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl genSmallR(char *curve, int *size)
{
   pairing_t pairing;
   element_t r;

   pairing_init_set_str(pairing, curve);
   element_init_Zr(r, pairing);
   element_random(r);
   *size = element_length_in_bytes(r);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, r);

   element_clear(r);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl calcU(char *curve, unsigned char *Pbyte, unsigned ↩
    char *rbyte, int *size)
{
   pairing_t pairing;
   element_t P, r, U;

   pairing_init_set_str(pairing, curve);
   element_init_G1(P, pairing);
   element_init_G1(U, pairing);
   element_init_Zr(r, pairing);
   element_from_bytes(r, rbyte);
   element_from_bytes(P, Pbyte);
   element_mul_zn(U, P, r);
   *size = element_length_in_bytes(U);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, U);

   element_clear(U);
   element_clear(P);
   element_clear(r);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl calcrQ(char *curve, unsigned char *Qbyte, unsigned ↩
    char *rbyte, int *size)
{
```

```
    pairing_t pairing;
    element_t Q, r, rQ;

    pairing_init_set_str(pairing, curve);
    element_init_G1(Q, pairing);
    element_init_G1(rQ, pairing);
    element_init_Zr(r, pairing);
    element_from_bytes(Q, Qbyte);
    element_from_bytes(r, rbyte);
    element_mul_zn(rQ, Q, r);
    *size = element_length_in_bytes(rQ);
    unsigned char *data = malloc(*size);
    element_to_bytes(data, rQ);

    element_clear(rQ);
    element_clear(Q);
    element_clear(r);
    pairing_clear(pairing);

    return data;
}

__declspec(dllexport) unsigned char * __cdecl calcV(char *curve, unsigned char *rbyte, unsigned ↩
    char *H1hash, int H1len, unsigned char *rQbyte, int *size)
{
    pairing_t pairing;
    element_t r, H1, rQ, rH1, V;

    pairing_init_set_str(pairing, curve);
    element_init_G1(H1, pairing);
    element_init_G1(rQ, pairing);
    element_init_G1(rH1, pairing);
    element_init_G1(V, pairing);
    element_init_Zr(r, pairing);
    element_from_hash(H1, H1hash, H1len);
    element_from_bytes(r, rbyte);
    element_from_bytes(rQ, rQbyte);
    element_mul_zn(rH1, H1, r);
    element_add(V, rH1, rQ);
    *size = element_length_in_bytes(V);
    unsigned char * data = malloc(*size);
    element_to_bytes(data, V);

    element_clear(r);
    element_clear(H1);
    element_clear(rQ);
    element_clear(rH1);
    element_clear(V);
    pairing_clear(pairing);

    return data;
}

__declspec(dllexport) unsigned char * __cdecl calcW1(char *curve, unsigned char *eQPpubbyte, ↩
    unsigned char *smallRbyte, unsigned char *bigRbyte, int *size)
{
    pairing_t pairing;
    element_t r, R, eQPpub, eQPpubr, W1;

    pairing_init_set_str(pairing, curve);
    element_init_Zr(r, pairing);
    element_init_GT(R, pairing);
    element_init_GT(eQPpub, pairing);
    element_init_GT(eQPpubr, pairing);
    element_init_GT(W1, pairing);
    element_from_bytes(eQPpub, eQPpubbyte);
    element_from_bytes(r, smallRbyte);
    element_from_bytes(R, bigRbyte);
    element_pow_zn(eQPpubr, eQPpub, r);
    element_mul(W1, eQPpubr, R);
```

```
    *size = element_length_in_bytes(W1);
    unsigned char *data = malloc(*size);
    element_to_bytes(data, W1);

    element_clear(r);
    element_clear(R);
    element_clear(eQPpub);
    element_clear(eQPpubr);
    element_clear(W1);
    pairing_clear(pairing);

    return data;
}


__declspec(dllexport) unsigned char * __cdecl decryptCalcR(char *curve, unsigned char *Ubyte, ↩
     unsigned char *pkbyte, unsigned char *Ppubbyte, unsigned char *Vbyte, unsigned char *W1byte, ↩
     int *size)
{
    pairing_t pairing;
    element_t U, pk, Ppub, V, W1, eUpk, ePpubV, division, R;

    pairing_init_set_str(pairing, curve);
    element_init_G1(U, pairing);
    element_init_G1(pk, pairing);
    element_init_G1(Ppub, pairing);
    element_init_G1(V, pairing);
    element_init_GT(W1, pairing);
    element_init_GT(eUpk, pairing);
    element_init_GT(ePpubV, pairing);
    element_init_GT(division, pairing);
    element_init_GT(R, pairing);
    element_from_bytes(U, Ubyte);
    element_from_bytes(pk, pkbyte);
    element_from_bytes(Ppub, Ppubbyte);
    element_from_bytes(V, Vbyte);
    element_from_bytes(W1, W1byte);

    element_pairing(eUpk, U, pk);
    element_pairing(ePpubV, Ppub, V);
    element_div(division, eUpk, ePpubV);
    element_mul(R, division, W1);

    *size = element_length_in_bytes(R);
    unsigned char * data = malloc(*size);
    element_to_bytes(data, R);

    element_clear(U);
    element_clear(pk);
    element_clear(Ppub);
    element_clear(V);
    element_clear(W1);
    element_clear(eUpk);
    element_clear(ePpubV);
    element_clear(division);
    element_clear(R);
    pairing_clear(pairing);

    return data;
}
```

A.3 *Wrapper C#* - Esquema *Paterson IBS*

A.3.1 sign.h

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
```

```
#include<malloc.h>
#include<pbc/pbc.h>

__declspec(dllexport) unsigned char * __cdecl generateSignr(char *curve, int *size);
__declspec(dllexport) unsigned char * __cdecl calcSignR(char *curve, unsigned char *rbyte, unsigned↩
     char *Pbyte, int *size);
__declspec(dllexport) unsigned char * __cdecl calcSignS(char *curve, unsigned char *rbyte, unsigned↩
     char *Pbyte, unsigned char *H2hash, int H2len, unsigned char *pkbyte, unsigned char *Rbyte, ↩
    int *size);
__declspec(dllexport) char __cdecl verifySign(char *curve, unsigned char *Pbyte, unsigned char *↩
    Ppubbyte, unsigned char *H2hash, int H2len, unsigned char *idHash, int idHashLen, unsigned ↩
    char *Rbyte, unsigned char *Sbyte);
```

A.3.2 sign.c

```
#include"sign.h"

__declspec(dllexport) unsigned char * __cdecl generateSignr(char *curve, int *size)
{
   //initialize pairing
   pairing_t pairing;

   //elements of algebraic structure
   element_t r;

   pairing_init_set_str(pairing, curve);
   element_init_Zr(r, pairing);

   element_random(r);

   *size = element_length_in_bytes(r);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, r);

   element_clear(r);
   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) unsigned char * __cdecl calcSignR(char *curve, unsigned char *rbyte, unsigned↩
     char *Pbyte, int *size)
{
   //initialize pairing
   pairing_t pairing;

   //elements of algebraic structure
   element_t P, r, R;

   pairing_init_set_str(pairing, curve);
   element_init_G1(P, pairing);
   element_init_G1(R, pairing);
   element_init_Zr(r, pairing);

   element_from_bytes(r, rbyte);
   element_from_bytes(P, Pbyte);

   element_mul_zn(R, P, k);

   *size = element_length_in_bytes(R);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, R);

   element_clear(P);
   element_clear(r);
   element_clear(R);
   pairing_clear(pairing);

   return data;
}
```

```
__declspec(dllexport) unsigned char * __cdecl calcSignS(char *curve, unsigned char *rbyte, unsigned↩
     char *Pbyte, unsigned char *H2hash, int H2len, unsigned char *pkbyte, unsigned char *Rbyte, ↩
    int *size)
{
   //initialize pairing
   pairing_t pairing;

   //elements of algebraic structure
   element_t P, r, R, S, H2, H2P, pk, pkH3R, H2PpkH3R;
   mpz_t H3R;

   pairing_init_set_str(pairing, curve);
   element_init_G1(P, pairing);
   element_init_G1(R, pairing);
   element_init_G1(S, pairing);
   element_init_G1(H2P, pairing);
   element_init_G1(pk, pairing);
   element_init_G1(pkH3R, pairing);
   element_init_G1(H2PpkH3R, pairing);
   element_init_Zr(r, pairing);
   element_init_Zr(H2, pairing);

   mpz_init(H3R);

   element_from_bytes(r, rbyte);
   element_from_bytes(P, Pbyte);
   element_from_bytes(R, Rbyte);
   element_from_bytes(pk, pkbyte);

   element_from_hash(H2, H2hash, H2len);

   element_mul_zn(H2P, P, H2);
   element_to_mpz(H3R, R);
   element_mul_mpz(pkH3R, pk, H3R);
   element_add(H2PpkH3R, pkH3R, H2P);
   element_invert(r, r);
   element_mul_zn(S, H2PpkH3R, r);

   *size = element_length_in_bytes(S);
   unsigned char *data = malloc(*size);
   element_to_bytes(data, S);

   element_clear(P);
   element_clear(r);
   element_clear(R);
   element_clear(S);
   element_clear(H2);
   element_clear(H2P);
   element_clear(pk);
   element_clear(pkH3R);
   element_clear(H2PpkH3R);

   pairing_clear(pairing);

   return data;
}

__declspec(dllexport) char __cdecl verifySign(char *curve, unsigned char *Pbyte, unsigned char *↩
     Ppubbyte, unsigned char *H2hash, int H2len, unsigned char *idHash, int idHashLen, unsigned ↩
     char *Rbyte, unsigned char *Sbyte)
{
   //initialize pairing
   pairing_t pairing;

   //elements of algebraic structure
   element_t P, Ppub, R, S, H2, ePP, ePPH2, Qid, ePpubQid, ePpubQidH3R, eRS, sigResult;
   mpz_t H3R;

   pairing_init_set_str(pairing, curve);
```

```
    element_init_G1(P, pairing);
    element_init_G1(Ppub, pairing);
    element_init_G1(R, pairing);
    element_init_G1(S, pairing);
    element_init_G1(Qid, pairing);
    element_init_Zr(H2, pairing);
    element_init_GT(ePP, pairing);
    element_init_GT(ePPH2, pairing);
    element_init_GT(ePpubQid, pairing);
    element_init_GT(ePpubQidH3R, pairing);
    element_init_GT(eRS, pairing);
    element_init_GT(sigResult, pairing);

    mpz_init(H3R);

    element_from_bytes(P, Pbyte);
    element_from_bytes(R, Rbyte);
    element_from_bytes(S, Sbyte);
    element_from_bytes(Ppub, Ppubbyte);

    element_from_hash(H2, H2hash, H2len);

    //element_mul_zn(H2P, P, H2);
    element_pairing(ePP, P, P);
    element_pow_zn(ePPH2, ePP, H2);
    element_from_hash(Qid, idHash, idHashLen);
    element_pairing(ePpubQid, Ppub, Qid);
    element_to_mpz(H3R, R);
    element_pow_mpz(ePpubQidH3R, ePpubQid, H3R);
    element_mul(sigResult, ePPH2, ePpubQidH3R);
    element_pairing(eRS, R, S);

    char c = "o";

    if (element_cmp(eRS, sigResult) == 0)
    {
        c = "t";
    }
    else
    {
        c = "f";
    }

    element_clear(P);
    element_clear(Ppub);
    element_clear(R);
    element_clear(S);
    element_clear(H2);
    element_clear(ePP);
    element_clear(ePPH2);
    element_clear(Qid);
    element_clear(ePpubQid);
    element_clear(ePpubQidH3R);
    element_clear(eRS);
    element_clear(sigResult);

    pairing_clear(pairing);

    return c;
}
```

A.4 Exemplo de utilização da biblioteca criada para *C#*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using IBE_Library;
using System.Runtime.InteropServices;
```

```
using System.IO;

namespace ClassTest
{
    class Program
    {
        static void Main(string[] args)
        {
            //parameters
            string curve = @"type a
q 9266299743659617491657496574960206183940628411508441714421989520488372068668 30
54746288175278969863442003388641695903994573348804300035235455669484555175 27
h 160011587607540055741654752809243579101406160487510427106979832859283189304664
r 579101793951763247864221588843498972747616122039952869713937648535669057781 77
exp2 255
exp1 243
sign1 1
sign0 1";

            byte[] S;
            byte[] P;
            byte[] Q;
            byte[] Ppub;
            byte[] QPpub;

            S = IBE.GenerateMasterKey(curve);
            P = IBE.GeneratePparam(curve);
            Q = IBE.GenerateQparam(curve);
            Ppub = IBE.CalculateTparam(curve, P, S);
            QPpub = IBE.CalculatePairQPpub(curve, Q, Ppub);

            //set identities
            string[] identities = { "identity1", "identity2"};

            //open file to encrypt input.txt
            MemoryStream fileIn = new MemoryStream();
            using (FileStream fs = File.OpenRead("input.txt"))
            {
                fs.CopyTo(fileIn);
            }

            //encrypt input.txt with identities array
            MemoryStream inCrypted = IBE.EncryptStream(identities, null, fileIn, "", curve, P, Q, ↩
                QPpub);

            //save encrypted file as "encrypted.ibc"
            FileStream fileOut1 = new FileStream("encrypted.ibc", FileMode.Create, FileAccess.Write↩
                );
            inCrypted.WriteTo(fileOut1);
            fileOut1.Close();

            //generate private key for "identity1"
            byte[] privateKey = IBE.GetPrivateKey("identity1", curve, S);

            //decrypt file and save result to output.txt
            MemoryStream deCrypted = IBE.DecryptStream("identity1", privateKey, inCrypted, curve, ↩
                Ppub);
            FileStream fileOut = new FileStream("output.txt", FileMode.Create, FileAccess.Write);

            deCrypted.WriteTo(fileOut);
        }
    }
}
```