

Faculdade de Engenharia da Universidade do Porto



**Aplicação de Selos Temporais em S/MIME para
WebMail**

Vitor Manuel Carvalho Barros

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Telecomunicações

Orientador: Prof. Manuel Ricardo
Co-Orientador: Prof. Jaime Dias

Julho de 2010

A Dissertação intitulada

“MÓDULO S/MIME PARA SISTEMA DE WEBMAIL”

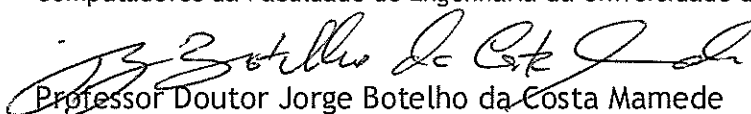
foi aprovada em provas realizadas em 20/Julho/2010

o júri



Presidente Professor Doutor Mário Jorge Moreira Leitão

Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto



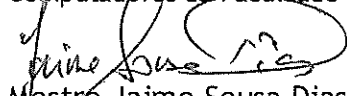
Professor Doutor Jorge Botelho da Costa Mamede

Professor Adjunto do Departamento de Engenharia Electrotécnica do Instituto Superior de Engenharia do Porto



Professor Doutor Manuel Alberto Pereira Ricardo

Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto (Orientador).



Mestre Jaime Sousa Dias

Investigador do INESC - PORTO (Co-Orientador).

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.



Autor - Vítor Manuel Carvalho Barros

Faculdade de Engenharia da Universidade do Porto

Resumo

Numa sociedade cada vez mais dependente das comunicações digitais, o email tem desempenhado um papel muito importante. Permite a transmissão de mensagens de forma semelhantes ao correio convencional. O uso de assinatura digital e cifragem fornecem ao email garantias, até então, reservadas ao correio convencional. Contudo, as implementações de segurança correntes não garantem a data e hora a que o email foi enviado.

Neste documento é proposta uma solução de *WebMail* que implementa o S/MIME e integra selos temporais ao nível da mensagem. Foi desenvolvido um protótipo que demonstra a utilização do email de forma segura, como uma possível alternativa ao correio convencional, sem que o utilizador tenha de lidar directamente com certificados e material criptográfico.

Página em branco

Abstract

In a society increasingly dependent on digital communications, email plays an important role. Allows the transmission of messages in a way similar to conventional mail. The use of digital signatures and encryption provides email security levels, until now, reserved to conventional mail. However, current security implementations do not guarantee the date and time the email was sent.

This document proposes a WebMail solution that implements S/MIME and includes timestamps to messages. A prototype was developed that demonstrates the use of email in a secure way, as a possible alternative to conventional mail, without the user having to deal directly with certificates and cryptographic material.

Página em branco

Agradecimentos

O meu sincero agradecimento ao Pedro Silva pela preciosa ajuda e paciência prestada do início ao fim da dissertação, sem ele não teria conseguido tudo o que foi aqui criado. Agradeço ao meu co-orientador Jaime Dias pelo apoio prestado e preocupação demonstrada ao longo de todo este ano. Agradeço ainda ao meu orientador, Manuel Ricardo, por se ter sempre mostrado compreensivo comigo quando nem sempre é fácil conciliar todas as tarefas que temos em mão.

Um grande e profundo obrigado à minha namorada e família pelo apoio. Gostaria de agradecer a todos os amigos que contribuíram para enriquecer este trabalho. Por último, agradeço a todos aqueles que não sendo mencionados, me apoiaram de uma forma desinteressada e amiga.

Vitor Barros

Página em branco

ÍNDICE

INTRODUÇÃO	1
1.1 TEMA E CONTEXTO	1
1.2 PROBLEMA	2
1.3 OBJECTIVO	2
1.4 ESTRUTURA DA DISSERTAÇÃO	3
DEFINIÇÃO DO PROBLEMA	5
2.1 CONTEXTUALIZAÇÃO DO PROBLEMA	5
2.2 <i>S/MIME</i> E <i>TIMESTAMPING</i>	6
2.3 <i>WEBMAIL</i>	8
2.4 CONCLUSÃO	10
ESTADO DA ARTE	11
3.1 NORMAS E SOFTWARE RELACIONADO	11
3.1.1 CERTIFICADOS X.509	11
3.1.2 <i>CERTIFICATE REVOCATION LIST (CRL)</i>	11
3.1.3 <i>ONLINE CERTIFICATE STATUS PROTOCOL (OCSP)</i>	12
3.1.4 <i>MULTIPURPOSE INTERNET MAIL EXTENSIONS (MIME)</i>	12
3.1.5 <i>SECURE/MULTIPURPOSE INTERNET MAIL EXTENSIONS (S/MIME)</i>	13
3.1.6 <i>TIMESTAMPING</i>	15
3.1.7 <i>OPENSSL</i>	16
3.1.8 <i>BOUNCYCASTLE</i>	16
3.1.9 <i>HORDE GROUPWARE WEBMAIL EDITION</i>	16
3.2 TRABALHOS RELACIONADOS	17
3.2.1 APLICAÇÕES DE <i>TIMESTAMPING</i>	17
3.2.2 <i>PLUG-IN</i> PARA <i>WEBMAIL</i>	18
3.2.3 CLIENTES DE EMAIL COM <i>TIMESTAMPING</i>	19
SOLUÇÃO PROPOSTA	25
4.1 ARQUITECTURA DA SOLUÇÃO	25
4.2 <i>WEBMAIL</i> COM <i>S/MIME</i>	27
4.3 <i>TIMESTAMPING</i> EM <i>S/MIME</i>	29
IMPLEMENTAÇÃO	33
5.1 MÓDULO DE CRIAÇÃO	34
5.1.1 VERIFICAÇÃO DE CERTIFICADOS	34

5.1.2	ASSINATURA	38
5.1.3	<i>TIMESTAMPING</i>	40
5.2	MÓDULO DE VERIFICAÇÃO	52
5.2.1	VERIFICAÇÃO DE CERTIFICADOS	52
5.2.2	<i>TIMESTAMPING</i>	52
5.3	INTEGRAÇÃO NO <i>HORDE</i> DO MÓDULO DE VERIFICAÇÃO	55
5.4	DISCUSSÃO	57
DEMONSTRAÇÃO E VALIDAÇÃO		59
6.1	MÓDULO DE CRIAÇÃO	59
6.1.1	VERIFICAÇÃO DE CERTIFICADOS	59
6.1.2	ASSINATURA	64
6.1.3	<i>TIMESTAMPING</i>	65
6.2	MÓDULO DE VERIFICAÇÃO	68
6.2.1	VERIFICAÇÃO DE CERTIFICADOS	68
6.2.2	<i>TIMESTAMPING</i>	68
6.3	INTEGRAÇÃO NO <i>HORDE</i> DO MÓDULO DE VERIFICAÇÃO	70
6.4	DISCUSSÃO	71
CONCLUSÕES		73
7.1	CONTRIBUIÇÕES E RESULTADOS	73
7.2	TRABALHO FUTURO	73
REFERÊNCIAS		75

Lista de figuras

Figura 2.1 - Troca de email simples com transporte seguro.....	6
Figura 2.2 - Exemplo de uma mensagem cifrada e assinada	7
Figura 2.3 - Alteração da data numa mensagem assinada	8
Figura 2.4 - Captura de imagem do <i>ThunderBird</i>	9
Figura 2.5 - Captura de imagem do <i>Gmail</i>	9
Figura 3.1 - Email MIME	13
Figura 3.2 - Cifragem e assinatura de mensagens, uso das chaves.....	14
Figura 3.3 - Processo de pedido de um <i>timestamp</i>	15
Figura 3.4 - Captura de ecrã do menu principal do <i>Horde</i>	17
Figura 3.5 - Aviso de não processamento de validação no <i>Gmail</i>	18
Figura 3.6 - Interface do <i>plug-in</i> para <i>Gmail</i>	19
Figura 3.7 - Abertura de mensagem original num email com MDDE	20
Figura 3.8 - <i>Link</i> para verificação no MDDE	21
Figura 3.9 - Instalação de software MDDE em ActiveX	21
Figura 3.10 - Abertura de MDDE no <i>browser</i>	22
Figura 3.11 - Selecção de ficheiro para verificação no MDDE	22
Figura 3.12 - Validação manual dos dados da mensagem MDDE	23
Figura 4.1 - Visão global do sistema de criação de mensagem	26
Figura 4.2 - Visão global do sistema de verificação de mensagem.....	26
Figura 4.3 - Aplicação de <i>timestamp</i> à passagem num MTA.....	27
Figura 4.4 - Exemplo de <i>WebMail</i> com S/MIME	28
Figura 4.5 - Integração do módulo de certificação num sistema <i>WebMail</i>	28
Figura 4.6 - Estrutura de um TSR.....	29
Figura 4.7 - Estrutura de um TST.....	30
Figura 4.8 - Anexo de TST a mensagem S/MIME	31
Figura 5.1 - Esquema ilustrativo da implementação.....	33
Figura 5.2 - Módulo de Criação	34
Figura 5.3 - Implementação da verificação por OCSP	35

Figura 5.4 - Implementação da verificação CRL	37
Figura 5.5 - Integração das verificações por OCSP e CRL	38
Figura 5.6 - Assinar mensagens	39
Figura 5.7 - Anexar o TST à mensagem S/MIME	51
Figura 5.8 - Módulo de Verificação	52
Figura 5.9 - Método de extracção de um TST a partir de um email	54
Figura 5.10 - Verificação do TST no módulo de Verificação.....	55
Figura 5.11 - Integração <i>Horde</i>	56
Figura 5.12 - Implementação do módulo de verificação no <i>view.php</i>	57
Figura 6.1 - Captura de pedido OCSP no <i>Wireshark</i>	60
Figura 6.2 - Captura da resposta OCSP no <i>WireShark</i>	61
Figura 6.3 - Verificação de uma mensagem OCSP.....	62
Figura 6.4 - Captura do pedido CRL no <i>WireShark</i>	63
Figura 6.5 - Captura da resposta CRL no <i>WireShark</i>	63
Figura 6.6 - Verificação de uma mensagem através de CRL.....	64
Figura 6.7 - Captura do resultado da assinatura de mensagens	64
Figura 6.8 - Captura da mensagem original e da mensagem assinada.....	65
Figura 6.9 - Captura do pedido <i>timestamp</i> à TSA no <i>WireShark</i>	66
Figura 6.10 - Captura da resposta enviada pela TSA no <i>WireShark</i>	66
Figura 6.11 - Análise do TST e validação do TSR recebido	67
Figura 6.12 - Anexação do TST à mensagem	67
Figura 6.13 - Pesquisa do TST na mensagem.....	68
Figura 6.14 - Verificação do TST no módulo de verificação	69
Figura 6.15 - Interface do <i>Horde</i> modificada	70
Figura 6.16 - Janela <i>pop-up</i> da verificação no <i>Horde</i>	71

Lista de tabelas

Tabela 5.1 - Funções relevantes do OpenSSL à assinatura de mensagens	39
Tabela 5.2 - <i>GETTERS</i> da classe <i>TimeStampRequest</i>	41
Tabela 5.3 - <i>SETTERS</i> da classe <i>TimeStampRequest</i>	42
Tabela 5.4 - Funções da classe <i>TimeStampToken</i>	48

Página em branco

Lista de abreviaturas e siglas

API	<i>Application Programming Interface</i>
CA	<i>Certificate Authority</i>
CMS	<i>Cryptographic Message Syntax</i>
CRL	<i>Certificate Revocation List</i>
DER	<i>Distinguished Encoding Rules</i>
GUI	<i>Graphical User Interface</i>
HSM	<i>Hardware Security Module</i>
HTTP	<i>HyperText Transfer Protocol</i>
IDE	<i>Integrated Development Environment</i>
IETF	<i>Internet Engineering Task Force</i>
IMAP	<i>Internet Message Access Protocol</i>
IMP	<i>Internet Messaging Program</i>
ITU-T	<i>Telecommunication Standardization Sector</i>
MAC	<i>Macintosh</i>
MIME	<i>Multipurpose Internet Mail Extensions</i>
MTA	<i>Mail Transfer Agent</i>
MUA	<i>Mail User Agent</i>
OCSP	<i>Online Certificate Status Protocol</i>
OID	<i>Object IDentifier</i>
PC	<i>Personal Computer</i>
PDA	<i>Personal Digital Assistant</i>
PEM	<i>Privacy-Enhanced Mail</i>
PGP	<i>Pretty Good Privacy</i>
PKI	<i>Public Key Infrastructure</i>
POP	<i>Post Office Protocol</i>
S/MIME	<i>Secure/Multipurpose Internet Mail Extensions</i>
SMTP	<i>Simple Mail Transfer Protocol</i>
SO	<i>Sistema Operativo</i>

SSL	<i>Secure Sockets Layer</i>
TLS	<i>Transport Layer Security</i>
TSP	<i>TimeStamp Protocol</i>
TTP	<i>Trusted Third Party</i>
URL	<i>Uniform Resource Locator</i>

Capítulo 1

Introdução

Este capítulo fornece uma visão global do trabalho desenvolvido e reportado na presente dissertação. Após uma breve exposição do tema abordado é apresentada uma descrição do problema, são enumerados os objectivos e, por último, é apresentada a estrutura da dissertação.

Esta dissertação insere-se num projecto empresarial cujo objectivo é dotar um sistema de correio electrónico com validade semelhante à encontrada nos sistemas de correio convencional: pretende-se desenvolver uma solução integrada em *WebMail* que permita o envio e recepção de mensagens com garantias legais semelhantes às do correio registado. Esta solução terá que conter um sistema de assinatura e suporte de selos temporais criptográficos, sendo este o problema abordado nesta dissertação.

1.1 Tema e Contexto

O correio electrónico, usualmente abreviado por email¹, é um método de troca de mensagens através de redes digitais. Na sua forma mais básica, é efectuada sem recorrer a qualquer tipo de protecção, neste modelo não existem quaisquer garantias de privacidade ou autenticidade da mensagem, nem de confirmação da data/hora do envio desta. Actualmente existem mecanismos amplamente divulgados para gerar mensagens seguras, nomeadamente o *S/MIME* e o *OpenPGP*, que oferecem à mensagem integridade, autenticidade, não repúdio da origem e confidencialidade. Apesar destes mecanismos, a data/hora continua a ser um dos campos que não é devidamente autenticado. Para colmatar estas lacunas existem

¹ "email (não e-mail, Email ou eMail) é a forma oficialmente definida pelo Internet Engineering Task Force (IETF)"

mecanismos criptográficos para aplicação de selos temporais — daqui em diante referido como "*timestamps*" — que permitem garantir que a determinada data/hora um email foi enviado ou passou por um *Mail Transfer Agent* (MTA).

Embora existam soluções de segurança em email, os utilizadores tendem a não adoptá-las essencialmente devido à sua falta de transparência. Na perspectiva do utilizador final, a usabilidade de um sistema tem um peso superior à segurança deste em sistemas não críticos [1]. Com o aumento da mobilidade dos utilizadores, quer em termos de localização, quer no acesso através de diferentes terminais, este problema torna-se mais relevante. Uma das formas de minimizar a falta de transparência em cenários de mobilidade passa pelo recurso à centralização de serviços. Opta-se cada vez mais por soluções *Web* para gestão de serviços devido à sua grande compatibilidade com qualquer tipo de terminais (PCs, MACs, PDAs, etc.). No panorama do email, as soluções de *WebMail* beneficiam destes trunfos que as tornam cada vez mais a solução adoptada neste cenário.

1.2 Problema

Que se conheça, não existe em *WebMail* uma solução integrada que permita a assinatura e cifragem de mensagens, acrescida de *timestamps* de forma a conferir a uma mensagem valor legal, especialmente em termos de não repúdio.

1.3 Objectivo

O trabalho consiste em estender uma solução integrada de segurança em *WebMail* para que suporte não só as funcionalidades típicas de segurança como assinar e cifrar, mas também o suporte de *timestamps* pelo Secure/Multipurpose Internet Mail Extensions (S/MIME) de forma a ser aferida a autenticidade de um email no espaço temporal. É importante desenvolver uma solução integrada para um serviço *WebMail* que permita assinar mensagens e aplicar *timestamps* criptográficos de forma a fornecer valor legal a um email, mantendo o sistema com o máximo de usabilidade e transparência para o utilizador final.

1.4 Estrutura da dissertação

Esta dissertação encontra-se organizada em 7 capítulos.

No capítulo seguinte, “Definição do Problema”, é descrito com maior detalhe o problema a resolver.

No terceiro capítulo, “Estado da Arte”, são introduzidas as ferramentas usadas e é efectuado um levantamento e análise de trabalho e conhecimento relacionado.

No quarto capítulo, “Solução Proposta”, desenvolve-se uma proposta de forma a resolver o problema colocado nesta dissertação.

No quinto capítulo, “Implementação”, explica-se todo o processo de implementação realizado e soluções adoptadas na criação do protótipo.

No sexto capítulo, “Demonstração e Validação”, apresentam-se os resultados e validações do trabalho desenvolvido.

No sétimo capítulo, “Conclusões”, apresenta-se uma síntese das contribuições, os resultados obtidos e trabalho futuro.

Capítulo 2

Definição do Problema

Neste capítulo são explicados os mecanismos que permitem garantir segurança a vários níveis e os problemas das implementações actuais. Inicialmente é feita uma contextualização sendo depois abordados os problemas mais especificamente. No final são retiradas algumas conclusões.

2.1 Contextualização do problema

O email é um mecanismo amplamente usado para troca de mensagens. Continua a ser um dos protocolos mais usados na Internet. É usado tanto para comunicações pessoais como empresariais. O acesso a contas de email é efectuado de duas formas distintas: a utilização de clientes de email (como o *Thunderbird* ou *Outlook*) e o acesso por *WebMail* (*Horde*, *Gmail*, etc.).

A segurança do email pode ser dividida em duas componentes, o transporte de mensagens e a mensagem em si. Para o transporte existem mecanismos amplamente divulgados e usados que fornecem segurança, nomeadamente o SSL/TLS. Na Figura 2.1 um diagrama demonstra o processo de troca de um email simples com recurso a transporte seguro. Uma mensagem enviada neste formato apesar de ter a transacção segura, não tem valor legal e não tem segurança em pontos intermédios. A quase totalidade das transacções de correio electrónico é feita neste formato. Quanto à segurança ao nível da mensagem, existem duas soluções que garantem integridade, autenticidade, não repúdio da origem e confidencialidade: o S/MIME e o OpenPGP.

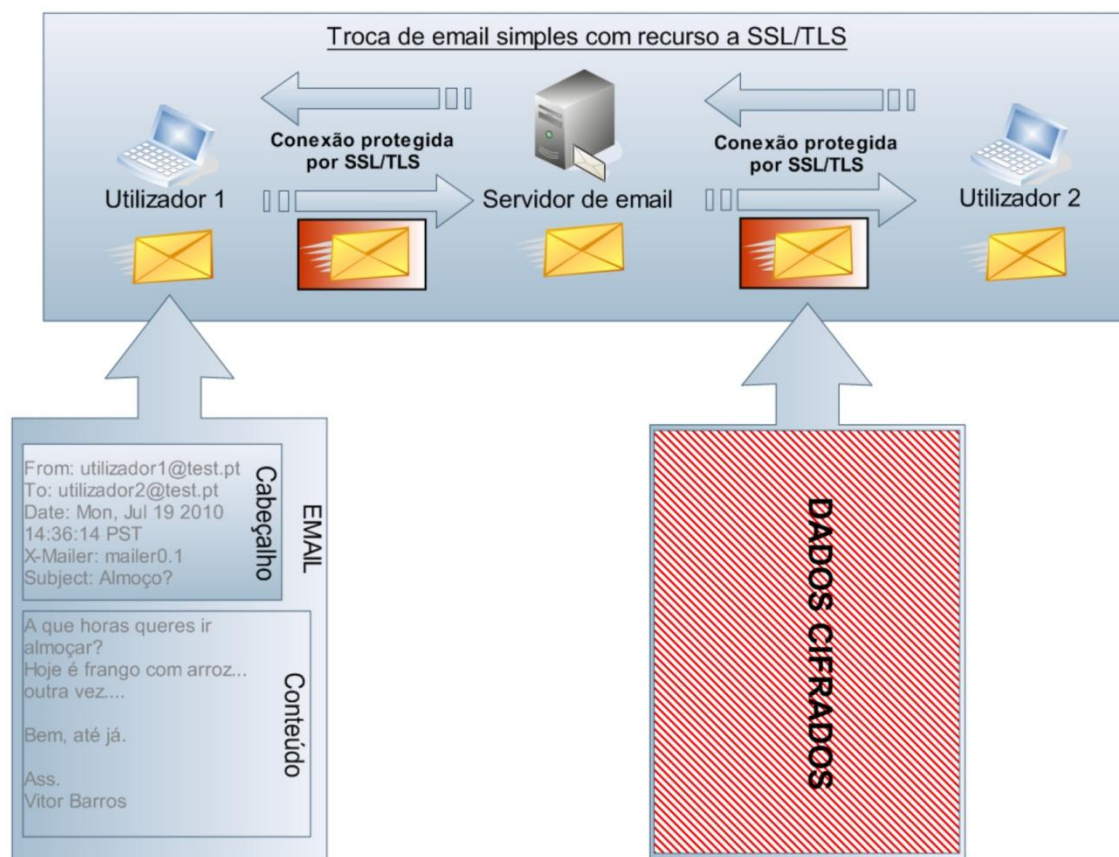


Figura 2.1 - Troca de email simples com transporte seguro

A data de envio ou passagem por um MTA de uma mensagem não pode ser irrefutavelmente provada recorrendo somente ao S/MIME ou ao PGP, é necessária uma terceira entidade que emita um *timestamp* para este efeito.

2.2 S/MIME e *Timestamping*

Apesar do S/MIME e o OpenPGP poderem garantir integridade, autenticidade, não repúdio da origem e confidencialidade, o S/MIME já vem instalado nos clientes de email (MUA) e é baseado em certificados X.509, os quais são os únicos com reconhecimento legal quando emitidos por autoridades de certificação (*Certificate Authority* - CA) comerciais. Esta foi a principal razão que levou o S/MIME a ser escolhido para o projecto empresarial, e consequentemente a ser considerado nesta dissertação.

Na Figura 2.2 pode ver-se um exemplo de uma mensagem cifrada e assinada recorrendo ao S/MIME. A cifragem é feita da origem ao destino, garantindo que mesmo nos pontos intermédios a mensagem mantém o sigilo dos dados, o que não acontece com a cifragem de ligação apresentada na Figura 2.1. A assinatura permite garantir a autenticidade da mensagem, confirmar a sua integridade, e, por ser baseada em criptografia de chave pública, garantir o não repúdio da origem (signatário da mensagem).

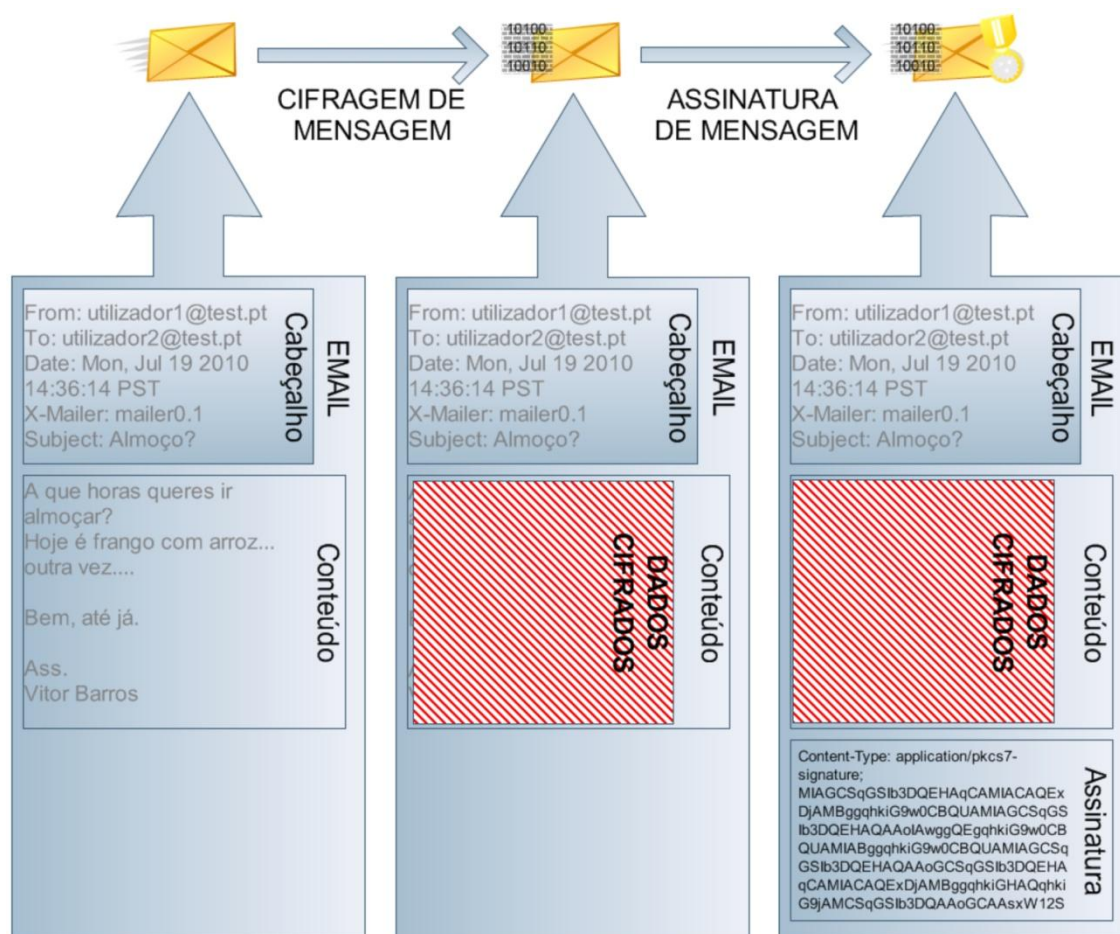


Figura 2.2 - Exemplo de uma mensagem cifrada e assinada

O S/MIME define um atributo, chamado “*Signing-Time*” [6], para transmitir a data em que a mensagem foi assinada. A data da assinatura é gerada pelo signatário da mensagem e, portanto, é tão confiável quanto o signatário. Um signatário malicioso pode alterar a data da mensagem assinada, sem que esta perca a validade da assinatura (ver Figura 2.3). Mesmo numa mensagem validada pelo utilizador final, a data não pode ser irrefutavelmente garantida como válida. Uma possível solução passa pelo recurso a *timestamps* emitidos por uma terceira entidade de confiança, uma *TimeStamping Authority* (TSA). O *timestamping* visa garantir que numa determinada data existiam determinados dados. No caso específico do email pode ser validada a data em que certo email foi enviado ou passou por um MTA. Uma limitação das implementações do S/MIME é a ausência de suporte nativo de *timestamping*.

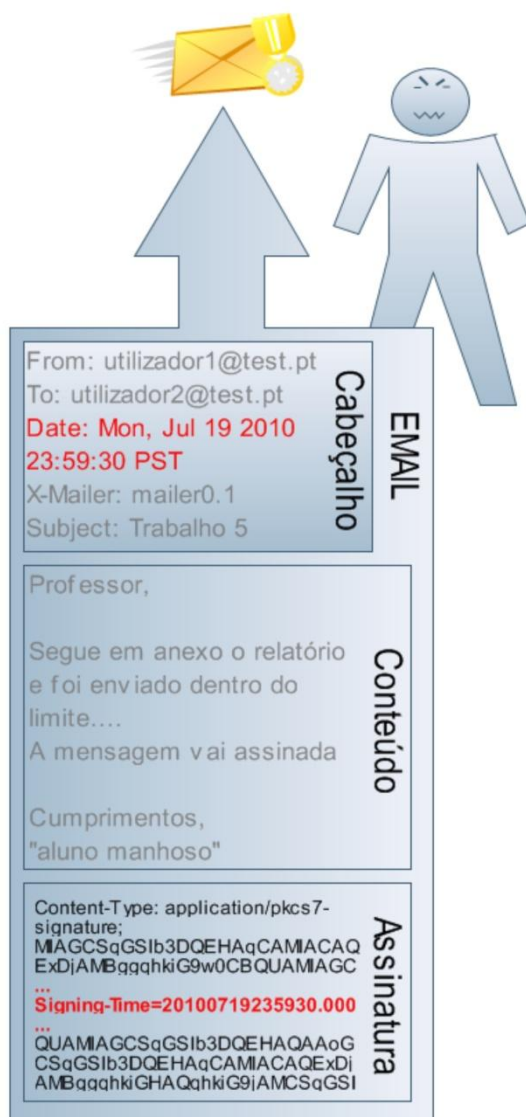


Figura 2.3 - Alteração da data numa mensagem assinada

2.3 WebMail

Apesar de os clientes de email (MUA) geralmente suportarem S/MIME, a utilização deste é muito reduzida pois o seu uso pressupõe algum conhecimento sobre certificação e manuseamento de certificados que não é comum à maioria dos utilizadores. Por outro lado, um sistema *WebMail* caracteriza-se pela sua versatilidade, e dado que o acesso é feito através de um *browser*, praticamente qualquer dispositivo com uma ligação à Internet pode aceder a uma conta de email, ler e enviar mensagens. A grande maioria dos sistemas *WebMail* não integra módulos de processamento de mensagens S/MIME. A falta de processamento nativo de *timestamps* afecta tanto os clientes de email como os sistemas *WebMail*. O *WebMail* devido ao factor mobilidade apresenta-se como a alternativa desejável para o

utilizador. Esta foi uma das razões que levou o *WebMail* a ser escolhido para o projecto empresarial, e consequentemente a ser considerado nesta dissertação.

Nas Figura 2.4 e Figura 2.5 pode ver-se um exemplo de um cliente de email e de um sistema *WebMail* respectivamente.

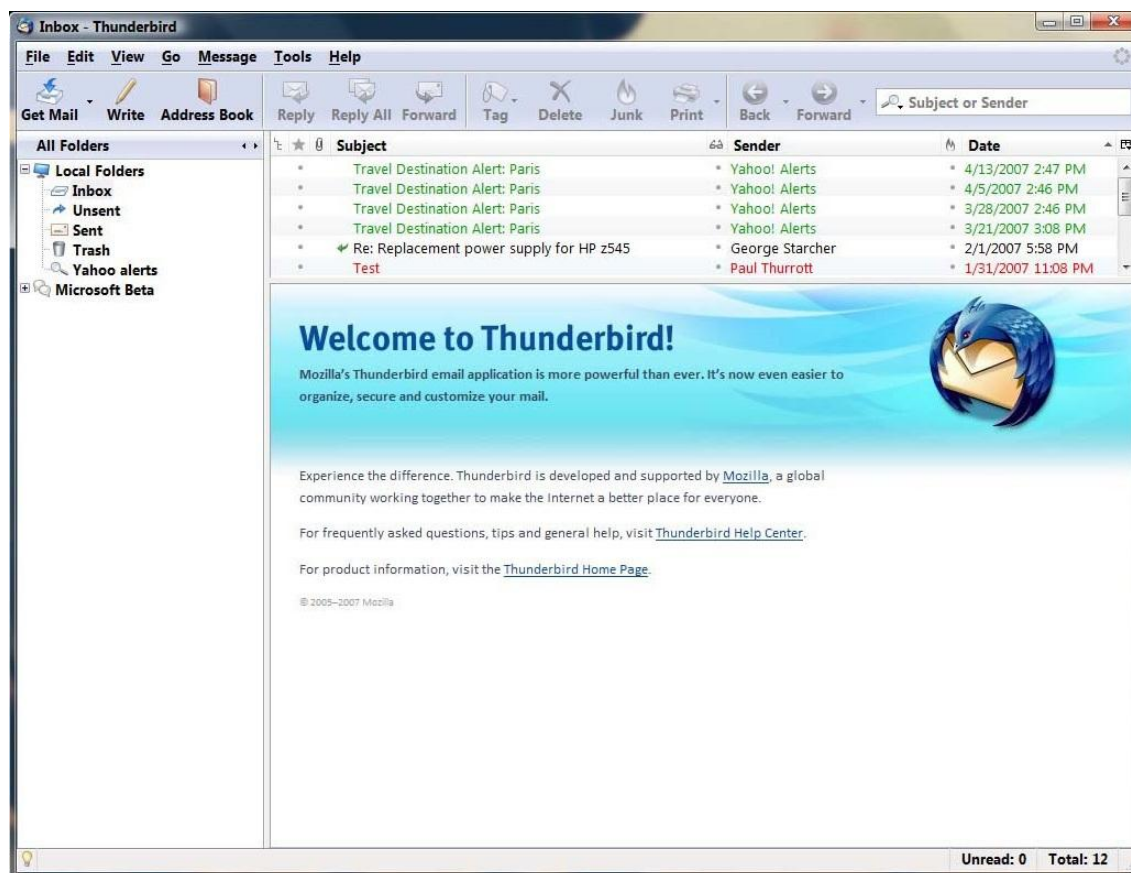


Figura 2.4 -Captura de imagem do *ThunderBird*

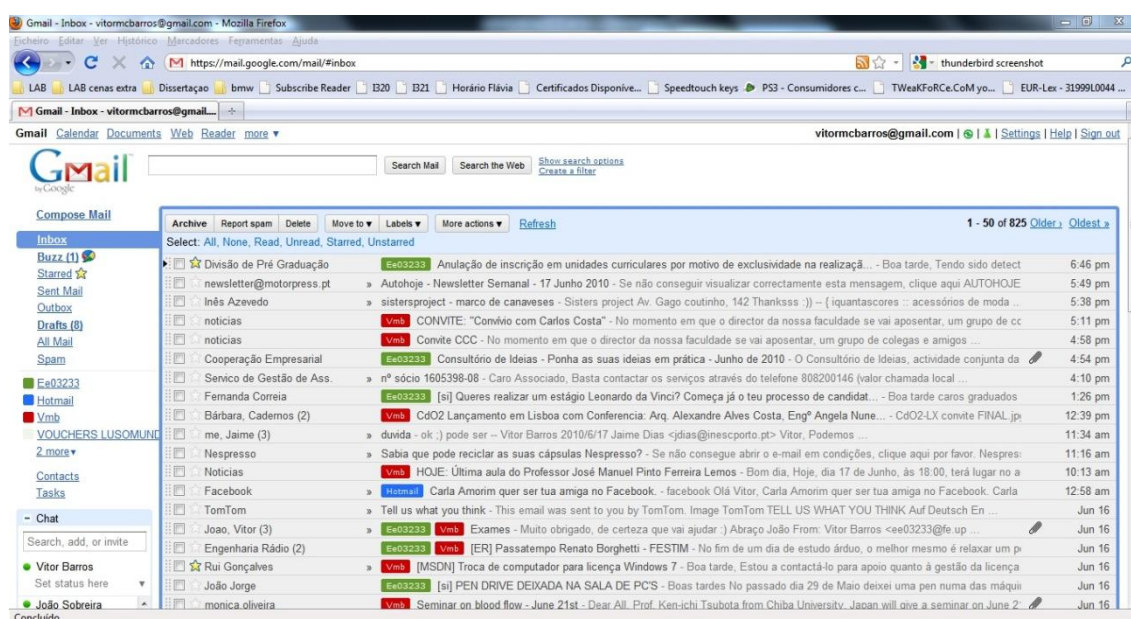


Figura 2.5 - Captura de imagem do *Gmail*

2.4 Conclusão

Neste capítulo foram identificados alguns problemas dos sistemas de email correntes:

- o email é usado na sua maioria sem recorrer a qualquer tipo de assinatura digital ou cifragem de dados;
- praticamente só os clientes de email suportam nativamente o S/MIME;
- o uso de assinaturas digitais e cifragem de dados pressupõe sempre um conhecimento sobre certificados e manuseamento destes, o que não é comum para a maioria dos utilizadores;
- mesmo usando as implementações de S/MIME, não é possível garantir o não repúdio da data de criação/envio de uma mensagem de email;
- não existe uma implementação de S/MIME que inclua suporte de *timestamps*.

Dos problemas identificados, a aplicação de *timestamps* em S/MIME e uso de certificação em *WebMail* são os únicos que carecem de aplicações. Os restantes problemas são derivados da usabilidade dos sistemas.

Capítulo 3

Estado da arte

Neste capítulo é realizada uma exposição sobre o estado da arte nos assuntos relacionados com a presente dissertação.

3.1 Normas e *software* relacionado

3.1.1 Certificados X.509

O X.509 é uma recomendação da *Telecommunication Standardization Sector* (ITU-T) que faz parte da série de recomendações X.500. O X.500 é uma norma que define um directório de serviço. Um directório de serviço consiste num ou mais servidores que mantêm uma base de dados distribuída de informação sobre utilizadores. O X.509 define uma *framework* para provisão de serviços de autenticação para utilizadores do directório X.500. No centro do esquema X.509 estão certificados de chave pública associados a cada utilizador. Assume-se que os certificados dos utilizadores são emitidos por uma CA e colocados no directório por esta ou pelos utilizadores. A versão mais recente dos certificados X.509 é a versão 3.

3.1.2 *Certificate Revocation List* (CRL)

Uma CRL é uma lista de certificados que foram revogados, e, portanto, não devem ser confiados. Cada CA deve manter uma lista de todos os certificados revogados mas não expirados, incluindo tanto os certificados emitidos para utilizadores como para outras CAs. Estas listas devem ser disponibilizadas no directório. Cada CRL colocada no directório é assinada pela CA e inclui: o nome da CA; a data de criação da lista; a data agendada para a emissão da próxima CRL; e uma entrada com o número de série e data de revogação de cada certificado revogado [4].

3.1.3 *Online Certificate Status Protocol (OCSP)*

O OCSP [16] é um protocolo usado para obter o estado de revogação de um certificado digital X.509. Foi criado como uma alternativa às CRLs, abordando especificamente alguns problemas associados ao uso de CRLs em *Public Key Infrastructures (PKI)* [18]. As mensagens transmitidas via OCSP são codificadas em ASN.1 e tipicamente transmitidas por HTTP. A troca de mensagens é feita num esquema de pedido/resposta.

3.1.4 *Multipurpose Internet Mail Extensions (MIME)*

O *Simple Mail Transfer Protocol (SMTP)* só suporta caracteres ASCII de 7-bit [5]. Isto significa que outros tipos de conteúdos não podem ser transmitidos por SMTP pois não podem ser correctamente representados. O formato *Multipurpose Internet Mail Extensions (MIME)* resolve as limitações do SMTP definindo mecanismos para enviar outros tipos de informação por email, incluindo conteúdos binários de 8-bits. Na Figura 3.1 pode ver-se um exemplo de uma mensagem MIME contendo várias partes [6]. Desta forma é possível transmitir qualquer tipo de dados desde que devidamente codificados. O MIME está normalizado nos *Request for Comments (RFCs)*: RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 e RFC 2049. É uma norma que expande o formato de email de forma a suportar:

- caracteres diferentes do ASCII;
- anexos;
- mensagens com múltiplas partes;

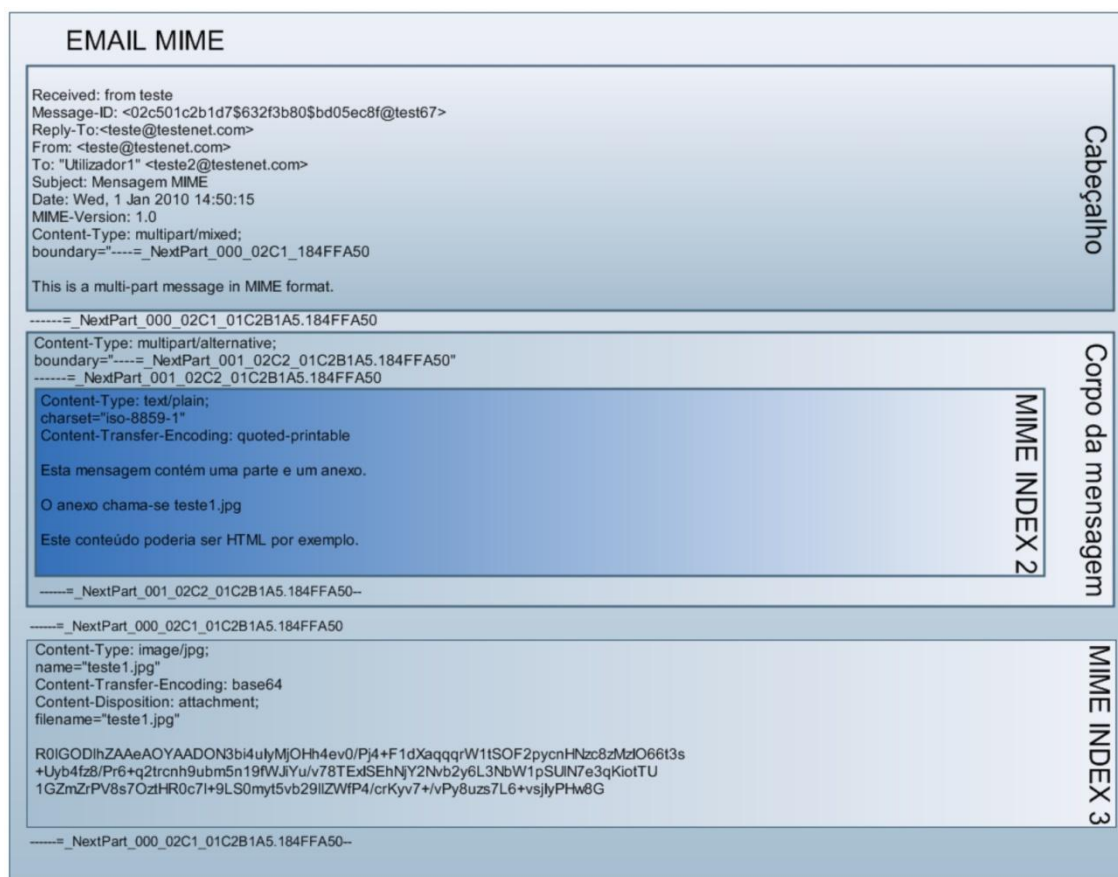


Figura 3.1 - Email MIME

3.1.5 Secure/Multipurpose Internet Mail Extensions (S/MIME)

O MIME não define mecanismos de segurança. Para colmatar esta lacuna foi especificado o *Secure/Multipurpose Internet Mail Extensions (S/MIME)*. O S/MIME foi criado originalmente pela empresa RSA Data Security [9]. Ratificado em Julho de 1999 pelo IETF [8], o S/MIME tornou-se uma norma, cujas especificações estão contidas nos RFC 2630 a 2633, 3369, 3370, 3850 e 3851. O S/MIME baseia-se em princípios de criptografia de chave pública. Permite cifrar as várias partes de uma mensagem MIME com a ajuda da chave pública do destinatário e assinar o corpo da mensagem com a chave privada do remetente. O processo para preparar uma mensagem cifrada é [4]:

1. Gerar uma chave de sessão aleatória para um algoritmo de cifragem simétrica (ex: RC2/40 ou 3DES).
2. Para cada destinatário, cifrar a chave de sessão com a chave pública de cada destinatário.

3. Para cada destinatário, preparar um bloco conhecido como *RecipientInfo* que contém um identificador do certificado X.509 dos destinatários, o algoritmo usado para cifrar a chave de sessão e a chave de sessão encriptada.

Consequentemente, apenas o(s) destinatário(s) pode(m) abrir o corpo da mensagem, com a ajuda da sua chave privada, o que assegura confidencialidade à mensagem recebida. A assinatura da mensagem assegura a integridade, autenticidade e o não repúdio de origem desta. A cifragem e a assinatura são processos que podem ser realizados independentemente. Na Figura 3.2 pode ver-se um diagrama simplificado explicativo dos mecanismos de cifragem e assinatura e como as respectivas chaves são usadas. O S/MIME é uma norma cuja especificação é baseada no *Cryptographic Message Syntax* (CMS), uma especificação IETF que é na sua maior parte idêntica ao PKCS#7 [5].

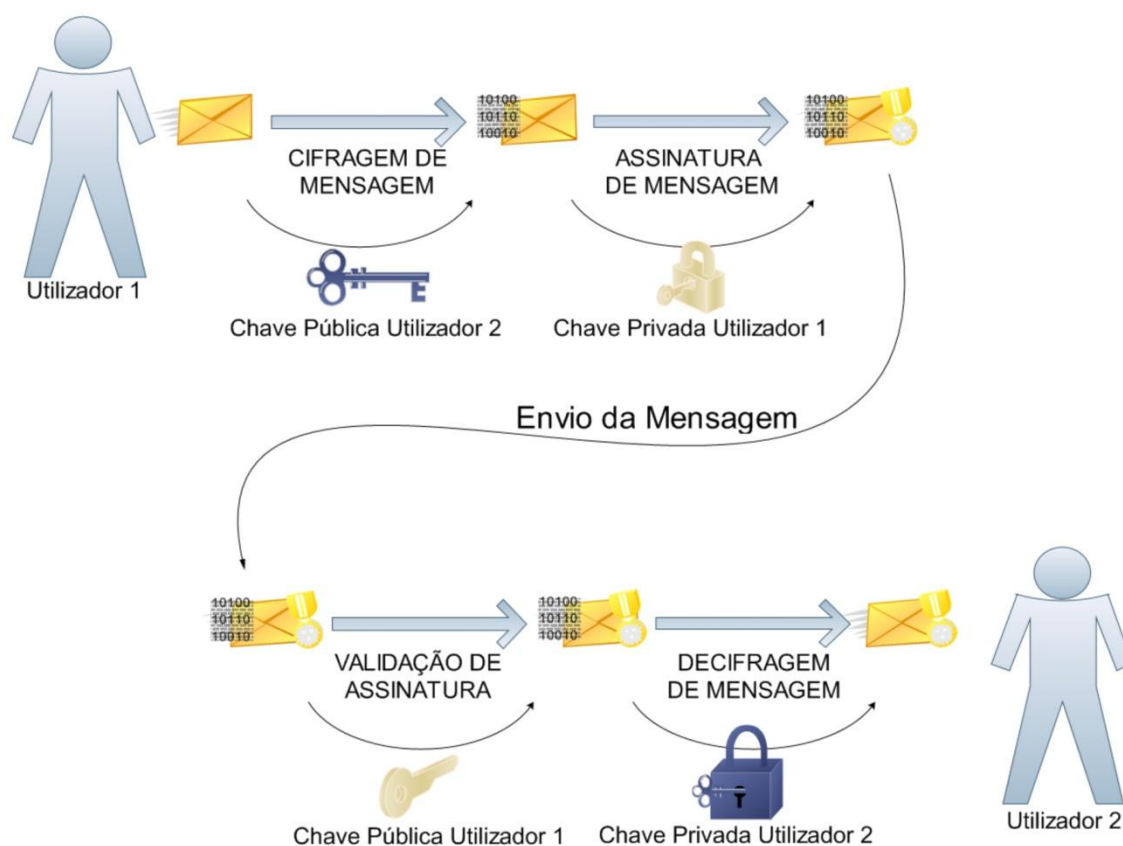


Figura 3.2 - Cifragem e assinatura de mensagens, uso das chaves

3.1.6 Timestamping

O *timestamping* é um serviço cujo interesse de aplicação pode ser encontrado nos mais diversos sistemas. O conceito de *timestamping* não é novo, quando Galileu e Robert Hooke queriam reclamar prioridade sobre uma descoberta mas não queriam publicá-la nesse momento, publicavam um anagrama cuja tradução fornecia o conteúdo original [10]. De acordo com o RFC 3161, um *timestamp* seguro é um *timestamp* emitido por uma terceira entidade de confiança (TTP), a agir na qualidade de autoridade de *timestamping* (TSA). O *timestamp* é usado para provar a existência de determinados dados antes de um certo ponto no tempo. Não existe a possibilidade de o proprietário alterar a data e hora para uma data e hora anterior à da existência dos dados. A técnica é baseada em assinaturas digitais e funções de resumo — daqui em diante chamadas *hash*². Primeiro, é calculada uma *hash*, *hash1*, a partir dos dados. A *hash1* é enviada para a TSA. A TSA concatena um *timestamp* com a *hash1* e calcula uma *hash2*. Esta *hash2* é por sua vez assinada digitalmente com a chave privada da TSA. A assinatura e o *timestamp* são enviados de volta para ao solicitante. Na Figura 3.3 pode ver-se este processo esquematizado.

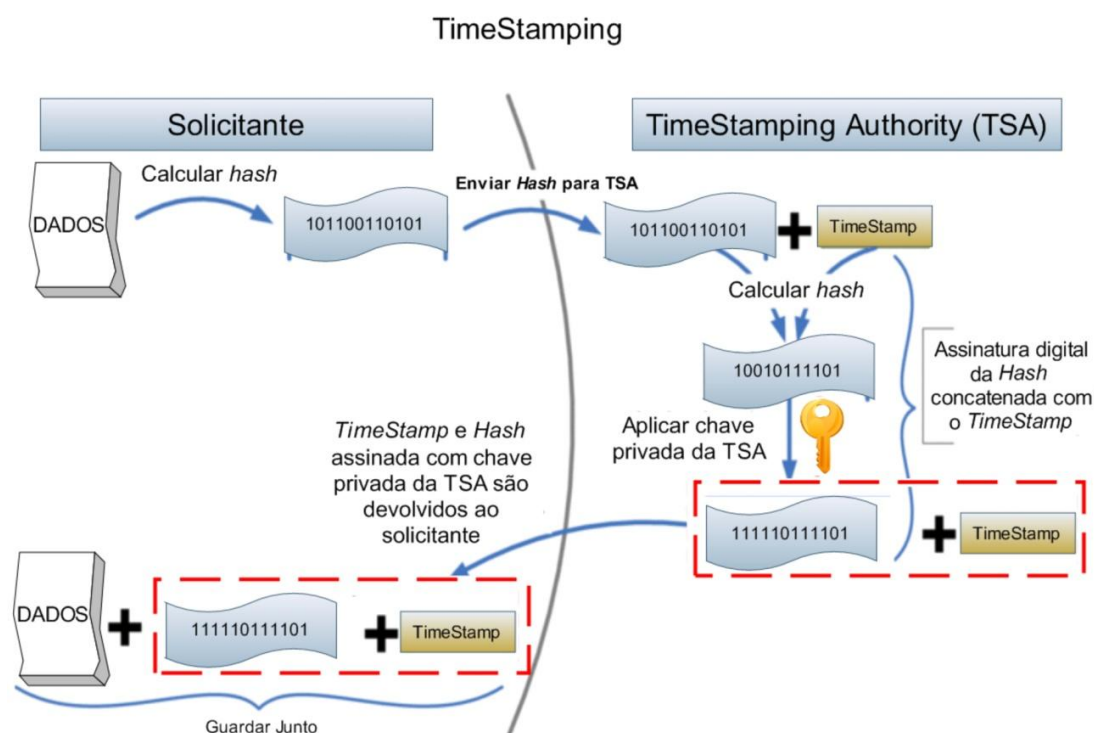


Figura 3.3 - Processo de pedido de um *timestamp*

² Uma função de resumo ou *hash* é um processo bem definido que converte um grande conjunto de dados num valor pequeno. Uma função de *hashing* é tão boa quanto a sua resistência a colisões, ou seja, cada conjunto de dados diferente, deve originar uma *hash* diferente, existindo uma probabilidade mínima de esta se repetir [4].

3.1.7 *OpenSSL*

O projecto *OpenSSL* é o resultado de um esforço colaborativo para desenvolver um *kit* de ferramentas que permita uma execução robusta do SSL e TLS. Dispõe de uma biblioteca de criptografia para uso geral. Esta é escrita em linguagem C e implementa muitos dos instrumentos criptográficos necessários ao desenvolvimento da solução. Este pacote pode ser dividido em duas vertentes: comandos que podem ser invocados directamente na consola com uma série de aplicações (ex. criar certificados) - e bibliotecas em C que podem ser usadas invocando funções disponíveis na API (*Application Programming Interface*). O *OpenSSL* é usado como interface entre o código desenvolvido e as estruturas de dados.

O código relativo ao processamento dos *timestamps* foi introduzido com a versão “1.0.0-beta1” do *OpenSSL*. Devido a esta inclusão ser recente, a documentação de apoio a estes módulos é bastante escassa.

3.1.8 *BouncyCastle*

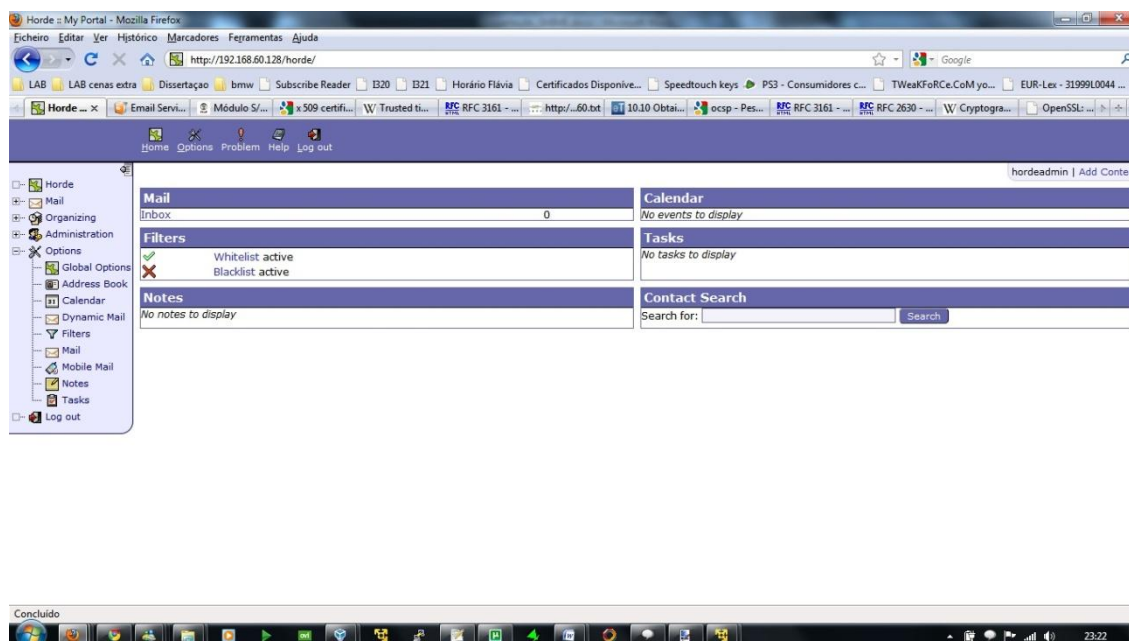
O *BouncyCastle* é uma *framework* que fornece várias bibliotecas, extensões e processadores em *Java* para realizar várias operações criptográficas [14]. Esta contém, entre outros, um gerador/processador para TSP de acordo com o RFC 3161. Permite efectuar várias operações de *timestamping* tal como se pretende no decorrer desta dissertação. Esta *framework* não é uma solução pronta a ser utilizada, pelo que o seu uso continua limitado às implementações que os programadores façam desta.

Tanto o *OpenSSL* como o *BouncyCastle* possuem processadores de *timestamps*, sendo que o primeiro está mais difundido.

3.1.9 *Horde Groupware WebMail Edition*

O *Horde Groupware Webmail* permite ler, enviar e organizar mensagens de email e gerir e partilhar calendários, contactos, tarefas e notas. Agrega as aplicações: *IMP*, *Ingo*, *Kronolith*, *Turba*, *Nag* e *Mnemo*. O *Horde* é uma *Framework* de aplicações *Web* baseada em PHP que oferece, entre outras, uma plataforma de email (*IMP*).

Para a aplicação dos módulos desenvolvidos num sistema WebMail foi escolhido o *Horde Groupware WebMail Edition 1.2.5* devido à sua natureza *opensource*, grande versatilidade e modularidade. Na aplicação dos módulos desenvolvidos ao *Horde*, foi efectuada alteração de código no módulo de email (*IMP*). Na Figura 3.4 pode ver-se uma captura do *Horde* em execução.

Figura 3.4 - Captura de ecrã do menu principal do *Horde*

Nas últimas versões do *Horde* está a ser introduzido o processamento de mensagens S/MIME e OpenPGP. Esta implementação ainda está numa fase inicial e pouco testada. Por exemplo, a documentação de apoio ao utilizador é bastante escassa e os processos de importação de certificados não são simples.

3.2 Trabalhos Relacionados

Não é conhecida nenhuma solução de *WebMail* que suporte S/MIME e *timestamps* e não foi encontrada nenhuma implementação de S/MIME com *timestamps*.

3.2.1 Aplicações de *Timestamping*

Vários serviços e aplicações podem beneficiar da aplicação de assinaturas temporais dos seus dados. O caso mais predominante será talvez o uso de *timestamping* em empresas para validar código e descobertas que possam vir a ser patentes, protegendo desta forma, por antecipação, os dados relativos. A COMODO, por exemplo, fornece serviços de assinatura de código para programadores informáticos. Este serviço permite que estes protejam o seu código contra alteração e corrupção. Como incrementação ao serviço base, têm ainda disponível o serviço de *timestamping* que valida *hashes* do código garantindo assim uma prova temporal da existência do código [11]. Este serviço funciona sob um modelo de compra de certificados para assinar o código e os *timestamps* são pagos adicionalmente.

3.2.2 Plug-in para WebMail

Existem sistemas de *WebMail* que suportam assinatura e cifragem recorrendo a *plug-ins*, mas não foi encontrado nenhum caso que suportasse *timestamps*. O Gmail, por exemplo, é um serviço de *WebMail* que não suporta cifragem nem assinatura, no entanto através da instalação de um *plug-in* é possível efectuar essas operações. Este tem que ser instalado no *browser*. Apresenta os mesmos problemas dos clientes de email, a gestão de certificados é feita localmente. Na versão mais recente à data da escrita desta dissertação, os certificados não são verificados (Figura 3.5).

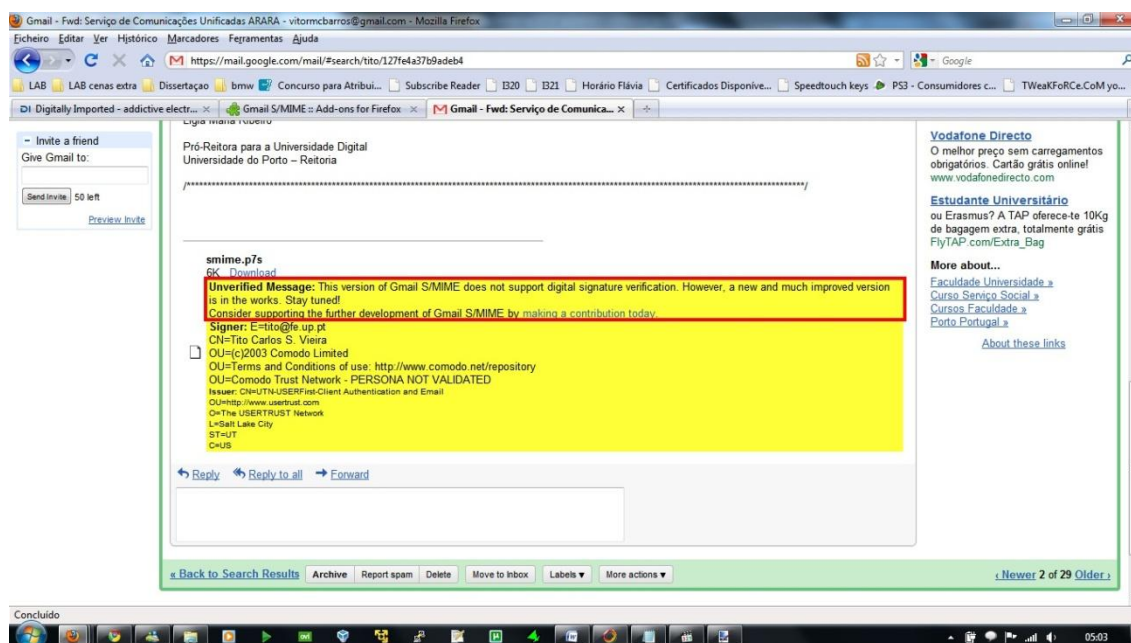


Figura 3.5 - Aviso de não processamento de validação no Gmail

Na Figura 3.6 pode ver-se uma captura de ecrã contendo a interface da criação de mensagens e as opções do *plug-in*. A interface foi alterada com a introdução de dois botões, um para assinar outro para cifrar. As opções permitem definir uma importação automática de certificados. Para sistemas *WebMail* é uma solução bastante aceitável na perspectiva de usabilidade. No entanto a gestão de certificados continua a ser executada localmente. O sistema não possui nenhuma forma de *timestamping*.

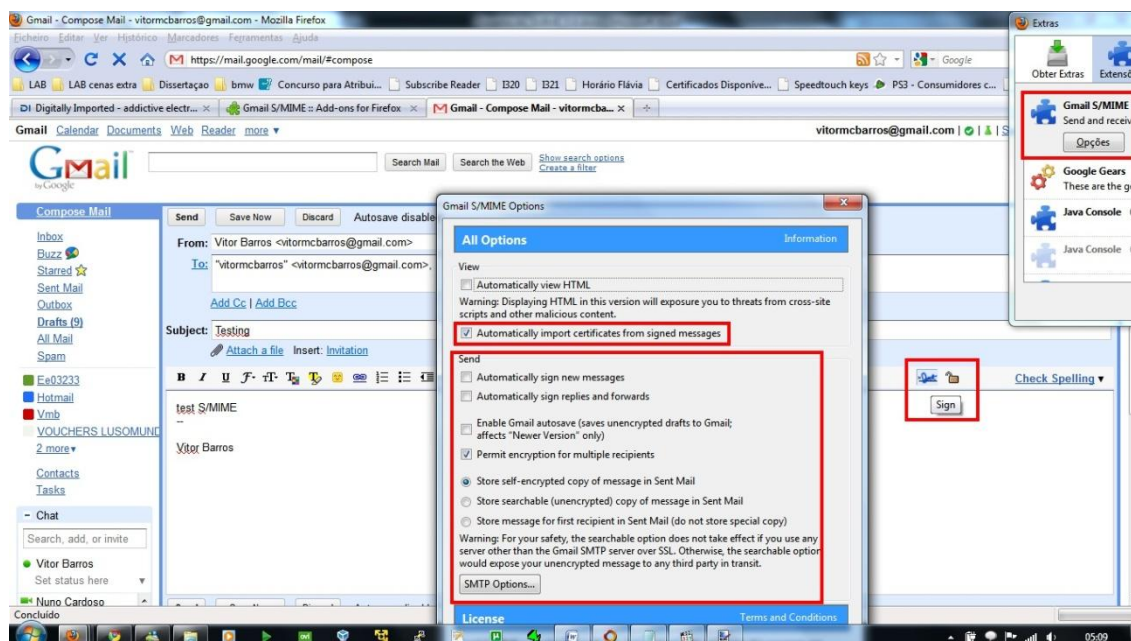


Figura 3.6 - Interface do *plug-in* para Gmail

3.2.3 Clientes de email com *Timestamping*

Existem clientes de email que suportam assinatura e *timestamps*, estes dependem da instalação de um *plug-in* e não são baseados em S/MIME. A Marca Do Dia Electrónica (MDDE), por exemplo, é um serviço concebido pelos CTT e pela MULTICERT, que possibilita a utilização do correio electrónico com segurança e fiabilidade. O serviço consiste na atribuição de um *timestamp* nos envios por correio electrónico, que verifica a veracidade da data e hora de envio, a integridade e assegura o não repúdio do conteúdo. O *timestamp* é emitido por uma terceira entidade de confiança e independente, e contém informação caracterizadora da transacção que permite aos remetentes e destinatários comprovarem a data e hora do envio electrónico, assim como a integridade do assunto, destinatários, corpo principal da mensagem e dos documentos anexos. Neste momento, o serviço apenas está disponível a utilizadores autorizados pela Ordem dos Advogados (Advogados inscritos na Ordem). Está previsto, a curto prazo, o desenvolvimento de novas soluções que permitam a utilização deste serviço por outras entidades e por qualquer utilizador [12].

A usabilidade do sistema é explicada um pouco melhor recorrendo ao manual de validação disponível, do qual são retirados excertos de texto e imagens [13]. Nesta explicação é demonstrado o processo necessário para validar uma mensagem com *timestamp*.

“ (...)

3-Abertura da mensagem original, que segue como anexo do email com MDDE recebido. (Figura 3.7)

a) Verificar a existência de dois anexos na mensagem, a saber, ficheiro com a mensagem original (cujo título é idêntico ao da mensagem recebida, exceptuando o prefixo [MDDE], no caso do Microsoft Outlook; ou “mail.eml” nos restantes clientes de email) e ficheiro cuja extensão é “.mdde”, o qual contém o *timestamp*.

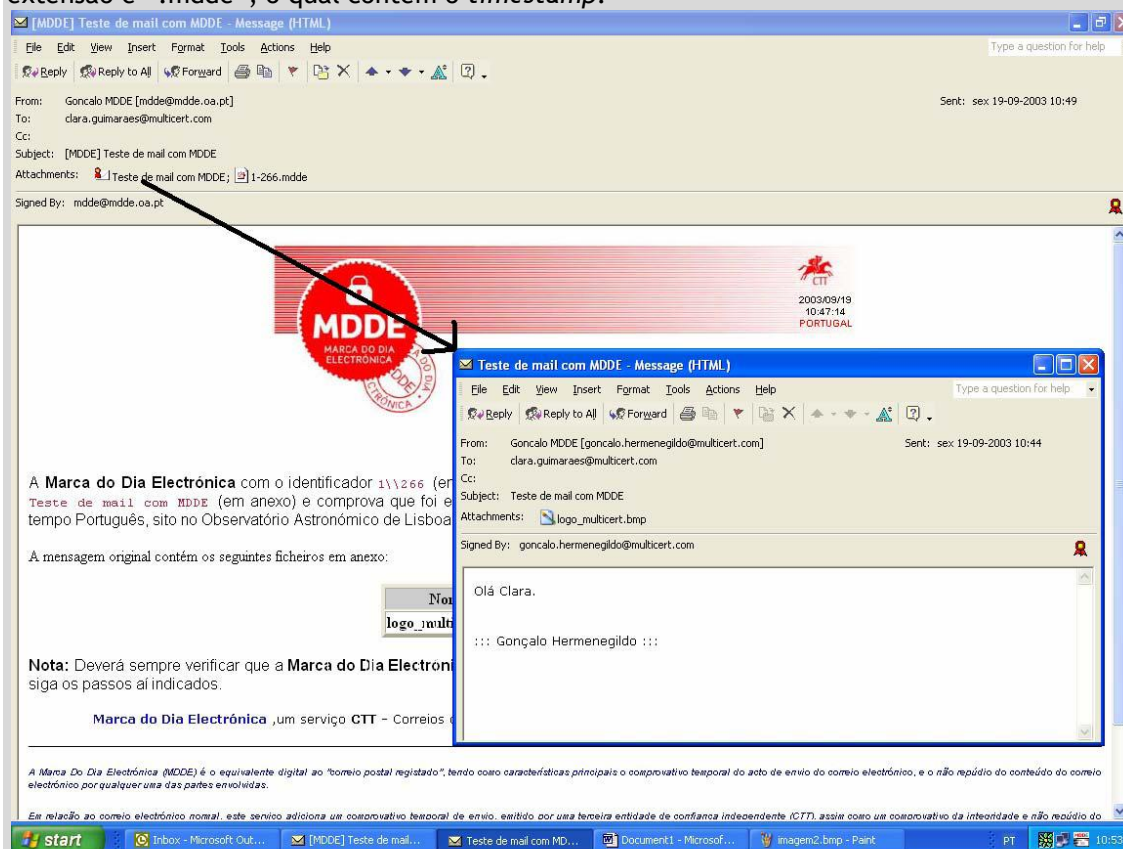


Figura 3.7 - Abertura de mensagem original num email com MDDE

4. Gravação da mensagem de e-mail com MDDE no disco local, no formato .msg (Microsoft Outlook) ou .eml (restantes clientes de e-mail).

(...)

5-Acesso ao *link* indicado no email com MDDE, para realização da validação (A validação é um passo opcional da prestação do serviço, pelo que a sua realização está essencialmente dependente do entendimento que os utilizadores fizerem da veracidade da mensagem recebida). (Figura 3.8)

(...)

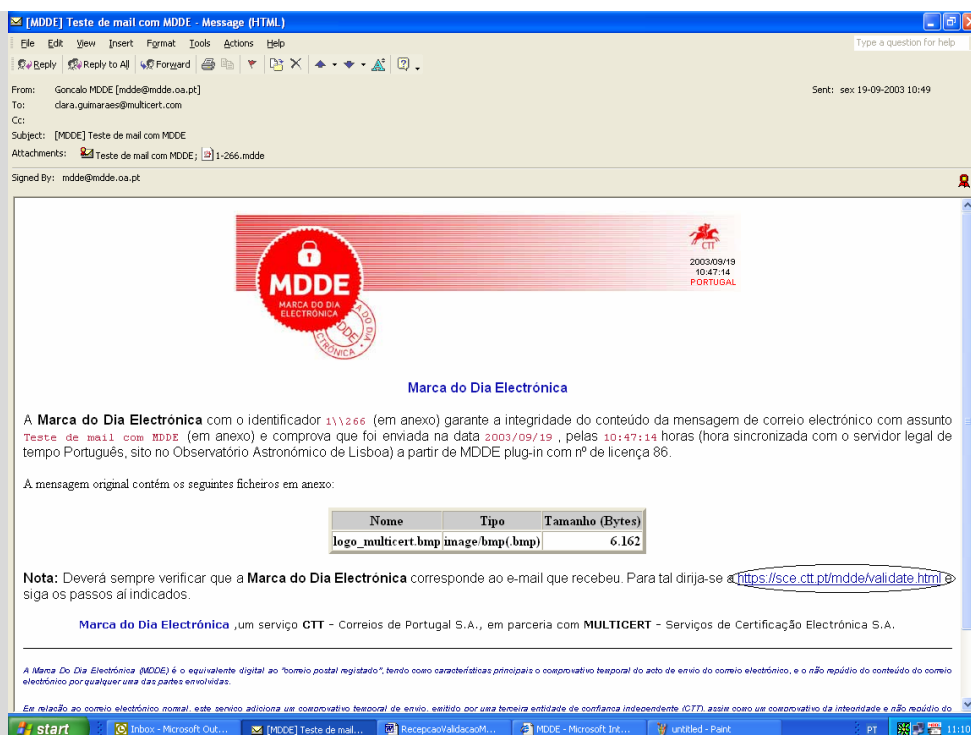


Figura 3.8 - Link para verificação no MDDE

6-Aceitar Instalação de Software MDDE, clicando em “SIM” (YES), para permitir fazer a validação (Figura 3.9).

(...)

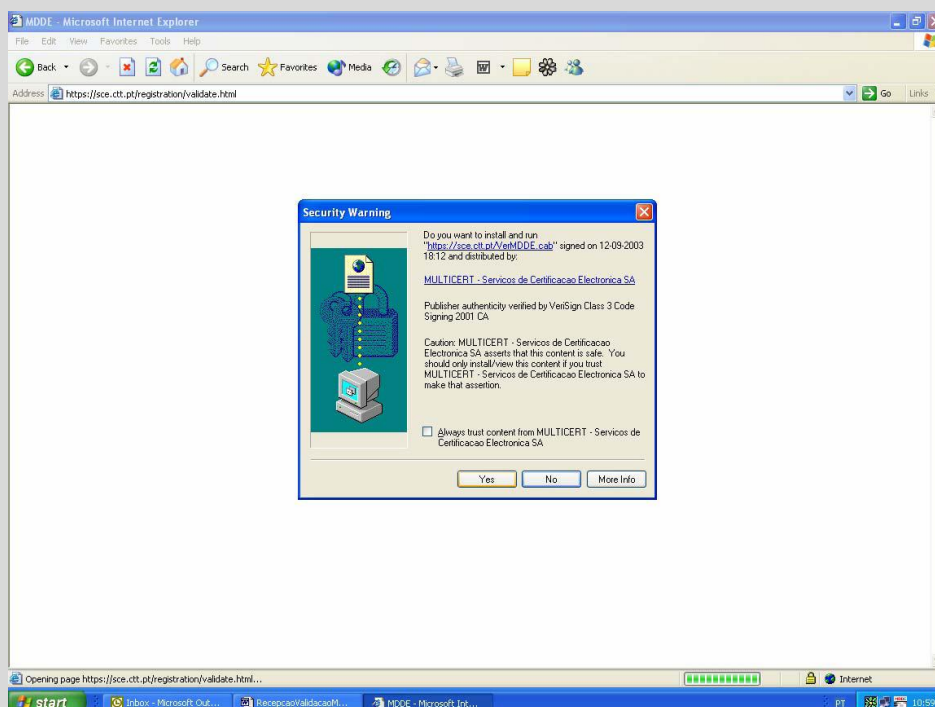


Figura 3.9 - Instalação de software MDDE em ActiveX

7-Clicar no botão “Procurar...” (*Browse...*) para seleccionar o ficheiro com a mensagem MDDE previamente gravado em disco. (Figura 3.10)

(...)

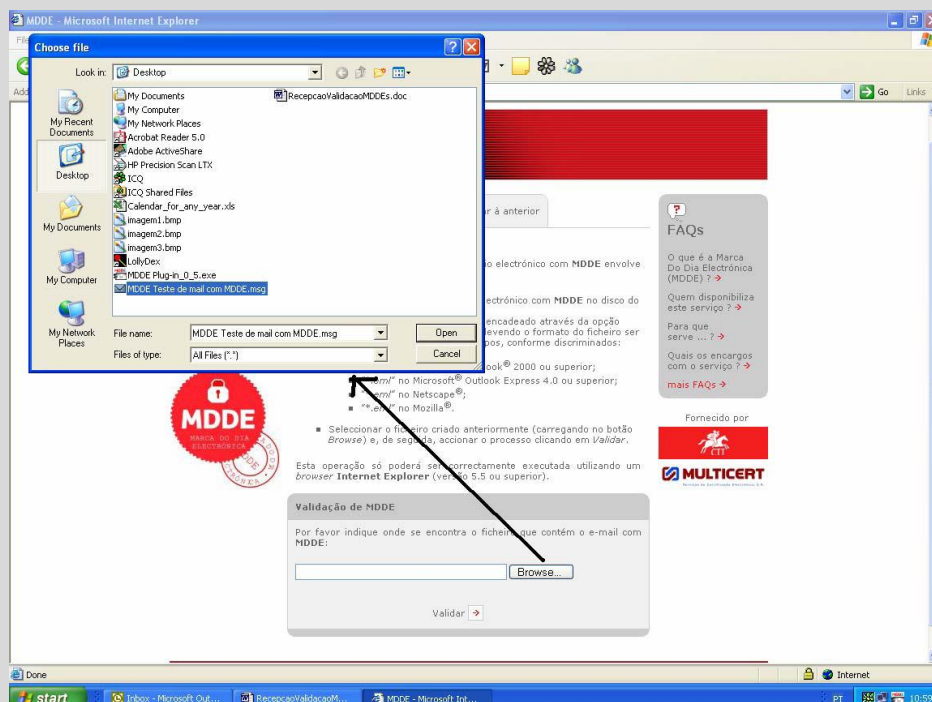


Figura 3.10 - Abertura de MDDE no browser

8-Clicar em “Validar”, depois do ficheiro seleccionado. (Figura 3.11)

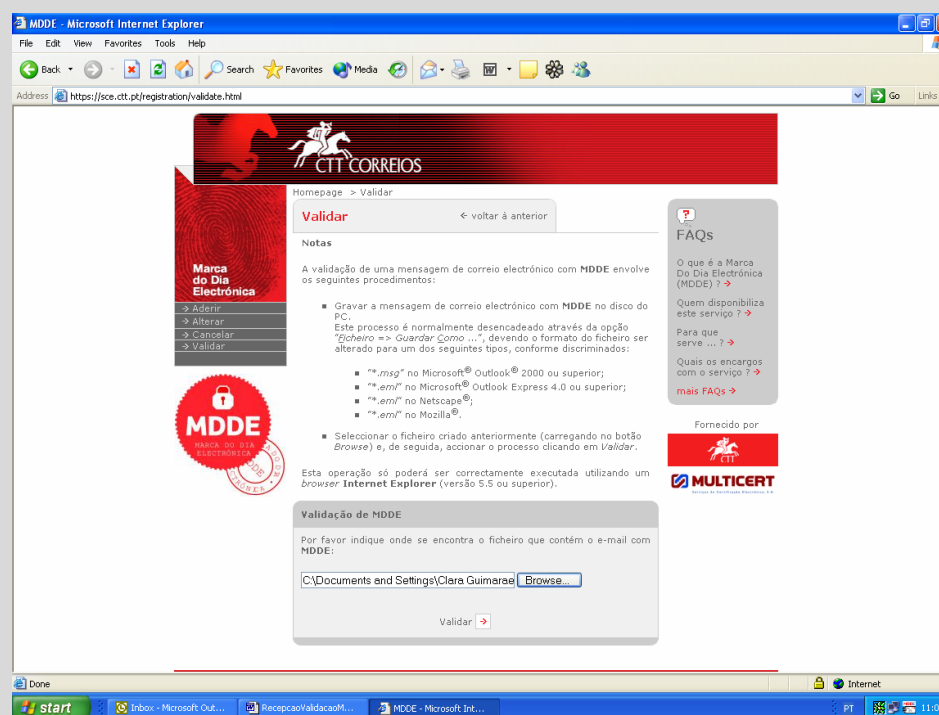


Figura 3.11 - Selecção de ficheiro para verificação no MDDE

9-Conferir na página apresentada todas as informações relacionadas com o envio (data e hora, emissor, destinatários, anexos, etc.). (Figura 3.12)



Figura 3.12 - Validação manual dos dados da mensagem MDDE

Para criação de mensagens, é necessária a instalação de um *plug-in* na máquina cliente. Este integra uma lista de contactos de forma a poder ser escolhido a que destinatários deve ser enviado o email através do MDDE. O *plug-in* ainda realiza a comunicação com os servidores de *timestamping* e faz o envio das mensagens para este.

O MDDE é um serviço de *timestamping* de mensagens, mas contém várias limitações:

- todas as mensagens recebidas terão a mesma origem tornando o processo de leitura mais demorado;
- a mensagem original não é apresentada directamente, esta vem em anexo;
- o processo de verificação envolve gravar a mensagem original localmente, o acesso a um *browser*, o envio da mensagem original e a verificação manual dos dados;
- está limitado a clientes de email, não podendo ser usado em *WebMail*;
- falta de portabilidade da solução, querendo usar o sistema num outro PC, implicaria ter o *plug-in* instalado.

Capítulo 4

Solução Proposta

Neste capítulo é apresentada uma solução que fornece integridade, autenticidade, não repúdio da origem e da data de envio dos emails. Esta solução é integrada em *WebMail* no contexto do projecto empresarial.

4.1 Arquitectura da Solução

De forma a resolver os problemas identificados anteriormente, a solução proposta consiste num sistema de *WebMail* que implementa o S/MIME e integra *timestamps*. Esta solução não depende da instalação de *software* na máquina cliente pelo que pode ser usada em qualquer terminal que disponha de um *browser*. A emissão de *timestamps* requer uma TSA. O processo de assinatura e aposição dos *timestamps* nos S/MIME é transparente para o utilizador.

Existem dois módulos para integrar a solução, um de criação e um de verificação. O módulo de criação é invocado durante o envio de uma mensagem e o de verificação quando uma mensagem é acedida pelo utilizador. Na Figura 4.1 pode ver-se uma visão global do mecanismo de criação de uma mensagem. Os métodos de assinatura e *timestamping* são aplicados após o cliente pedir o envio da mensagem garantindo que o *timestamp* contém a data de envio. O módulo de criação é invocado após o servidor de *WebMail* criar o contentor MIME da mensagem a enviar. Na Figura 4.2 pode ver-se uma visão global do sistema de verificação de mensagens. A verificação pode ser invocada sempre que uma mensagem é acedida ou a pedido do cliente. A validação oferece ao utilizador final garantias de integridade, autenticidade e não repúdio da origem e da data da mensagem.

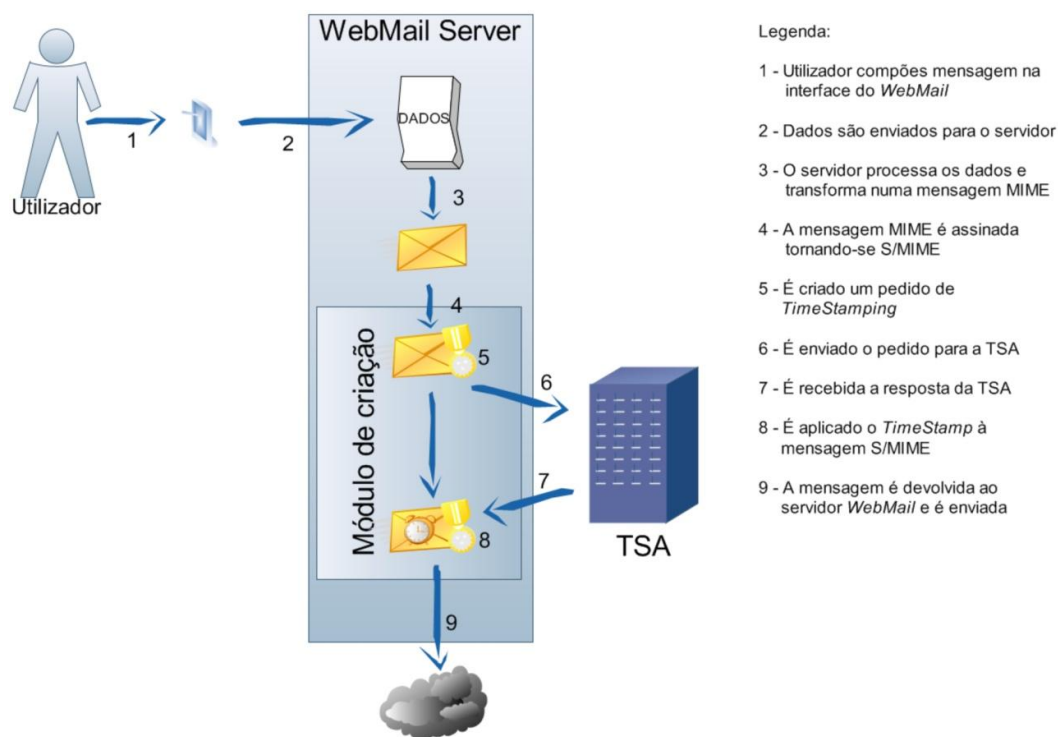


Figura 4.1 - Visão global do sistema de criação de mensagem

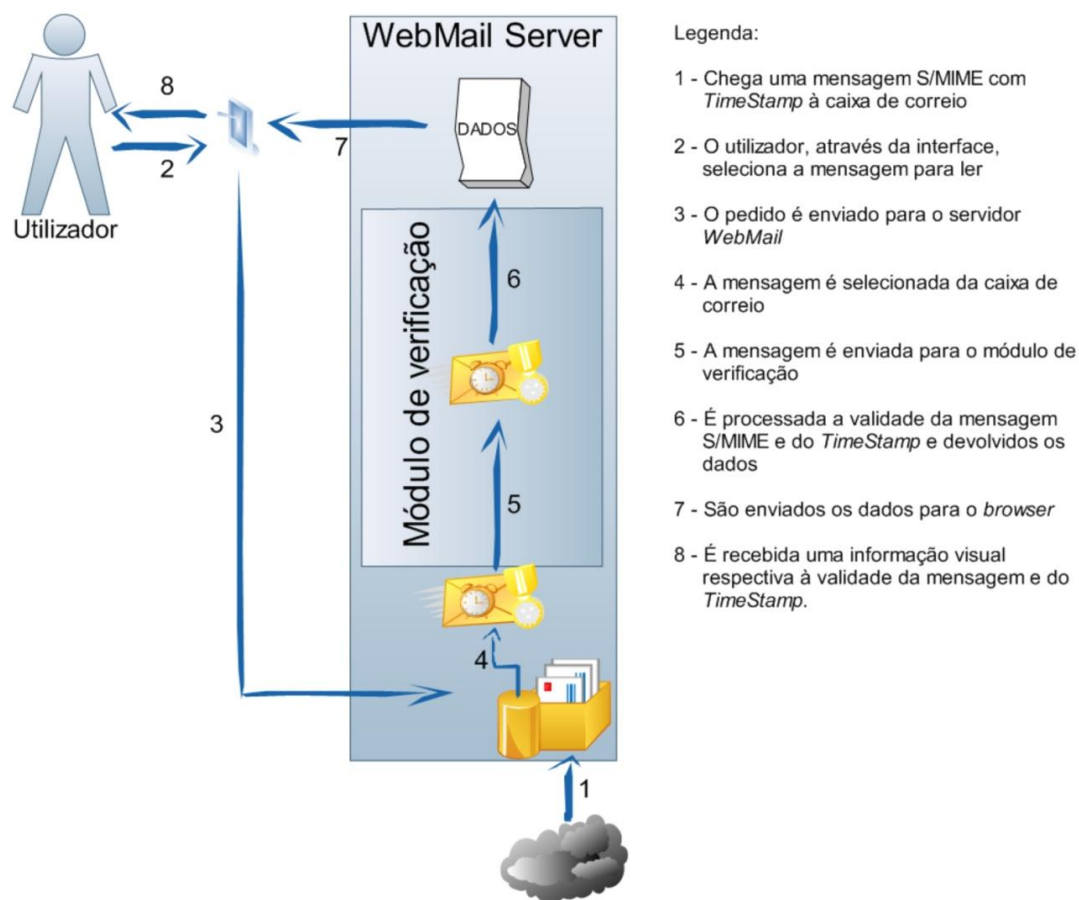


Figura 4.2 - Visão global do sistema de verificação de mensagem

Para mensagens enviadas a partir de clientes de email, a aplicação de *timestamps* pode ser efectuada à passagem de uma mensagem por um MTA, como se pode ver-se na Figura 4.3.

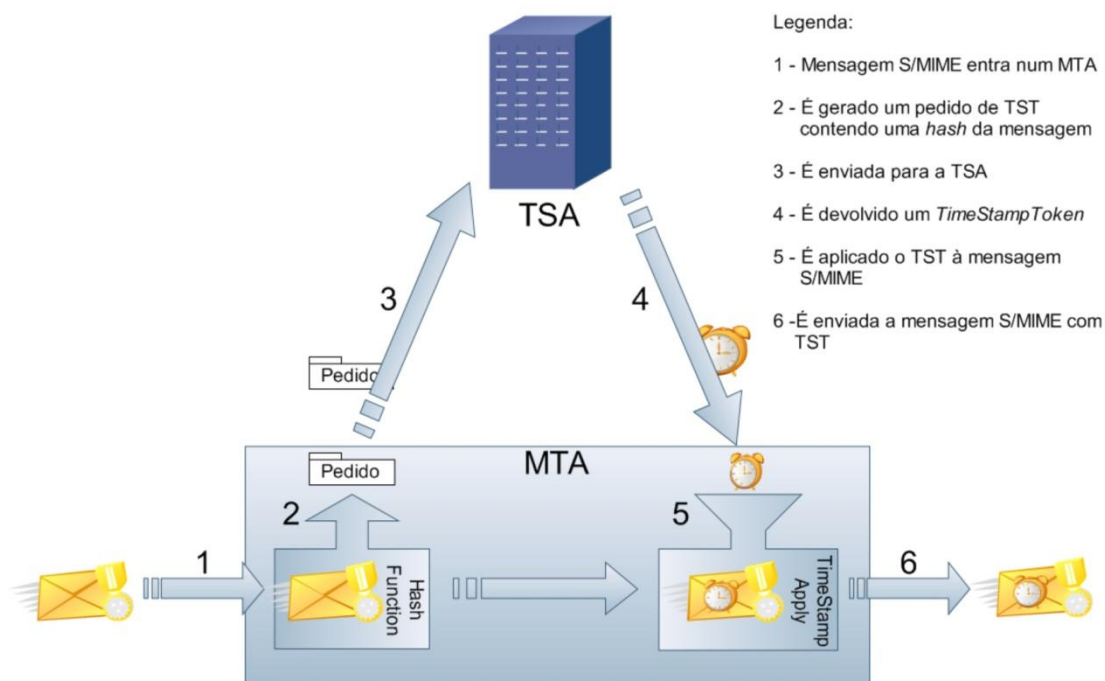


Figura 4.3 - Aplicação de *timestamp* à passagem num MTA

4.2 WebMail com S/MIME

O *WebMail* não inclui mecanismos de segurança nativos que permitam garantir integridade, autenticidade, confidencialidade e não repúdio de uma mensagem. Dado que o processamento das mensagens é efectuada do lado do servidor, há necessidade de fazer gestão de certificados e chaves privadas directamente no servidor *WebMail*. Devem ser obtidos os recursos necessários à gestão segura dos certificados X.509. Uma hipótese será utilizar módulos de *hardware* seguro (HSM) para a gestão de chaves privadas, no entanto, esta decisão sai fora do âmbito da dissertação ficando à responsabilidade da empresa onde o projecto se insere. A confiança nos certificados depende directamente da confiança depositada na organização que gere o *WebMail*. Assumindo que um servidor de *WebMail* gere certificados e chaves privadas, é possível a utilização de S/MIME. O *WebMail* passa a efectuar as operações de assinatura e cifragem das mensagens, serviços anteriormente suportados nas máquinas cliente. Na Figura 4.4 pode ver-se um exemplo de como funcionaria um sistema destes. A forma de implementação e obtenção de confiança para motivos legais por parte da organização, são temas que não são abordados nesta dissertação.

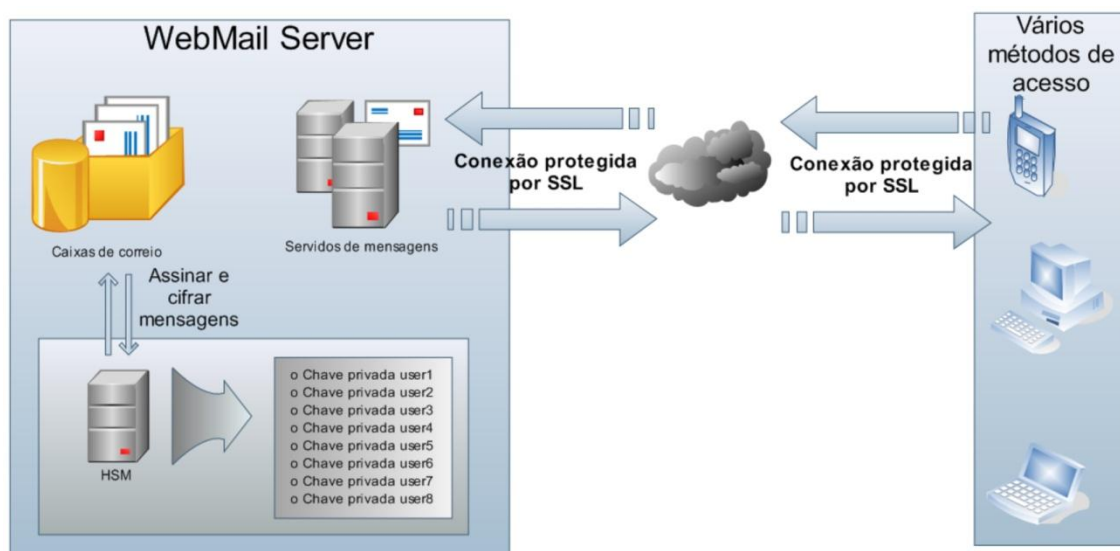


Figura 4.4 - Exemplo de *WebMail* com S/MIME

É necessário desenvolver módulos de *software* que executem operações de assinatura e cifragem de mensagens para um sistema *WebMail*. A solução de *software* proposta opera entre a *framework* e a interface gráfica (GUI) do serviço de email. Na Figura 4.5 pode ver-se um exemplo desta implementação num sistema *WebMail* com múltiplos serviços.

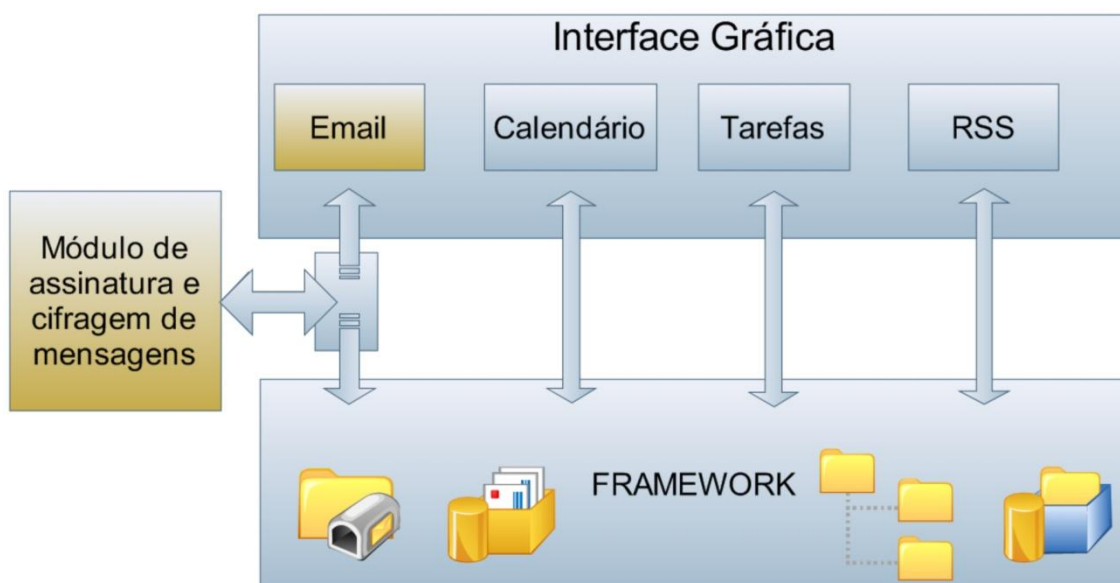


Figura 4.5 - Integração do módulo de certificação num sistema *WebMail*

Propõe-se a criação de módulos de *software* independentes a ser integrados em sistemas *WebMail*. A vantagem de uma solução a operar entre a GUI e a *framework* é a modularidade que se obtém com esta implementação. Este módulo poderá ser integrado em diferentes sistemas de *WebMail* com o mínimo de modificações. Acresce-se a vantagem de não ser alterada a *framework*, esta continuará a executar o trabalho de enviar e receber mensagens

da mesma forma que era executada antes. A mensagem será assinada e cifrada quando é retornada da *interface* para a *framework* e analisada quando é enviada da *Framework* para a GUI.

Para realizar esta implementação foi escolhido o *Horde*. Esta é uma escolha efectuada no âmbito do projecto empresarial no qual esta dissertação se insere. A grande modularidade do *Horde* e a sua natureza *opensource* contribuíram para esta decisão.

4.3 Timestamping em S/MIME

Um dos problemas apresentados pelos métodos de assinatura de mensagens é não se poder garantir a validade de uma mensagem no espaço temporal. Uma solução para validar dados numa dada data e tempo é recorrendo aos *timestamps*.

Dado que não existe uma implementação para a aplicação de *timestamps* em mensagens S/MIME, é necessário realizar uma. Esta foi feita seguindo as recomendações do RFC 3161 [15]. São retiradas várias referências para os parágrafos seguintes deste RFC.

Antes de se proceder à explicação de como aplicar os *timestamps*, terão que ser compreendidas algumas estruturas relevantes. Qualquer pedido de *timestamp* efectuado a uma TSA envolve a recepção de um TSR (*TimeStampReply*). Esta resposta vem estruturada de acordo com o diagrama da Figura 4.6.

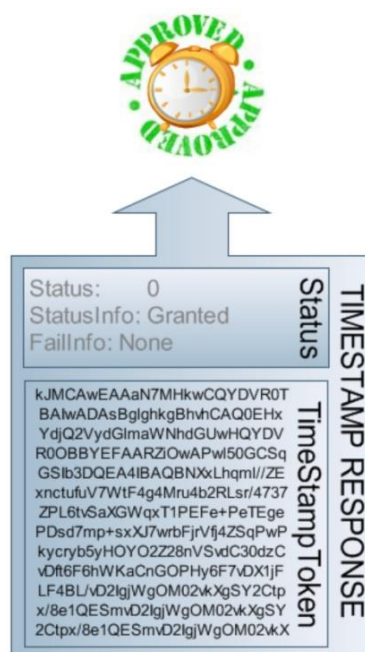


Figura 4.6 - Estrutura de um TSR

No campo “*status*” é enviada informação relativa à resposta, nomeadamente se o TST está presente e o código de erro no caso de este não estar.

O TST por sua vez tem a sua estrutura explicada no diagrama da Figura 4.7.

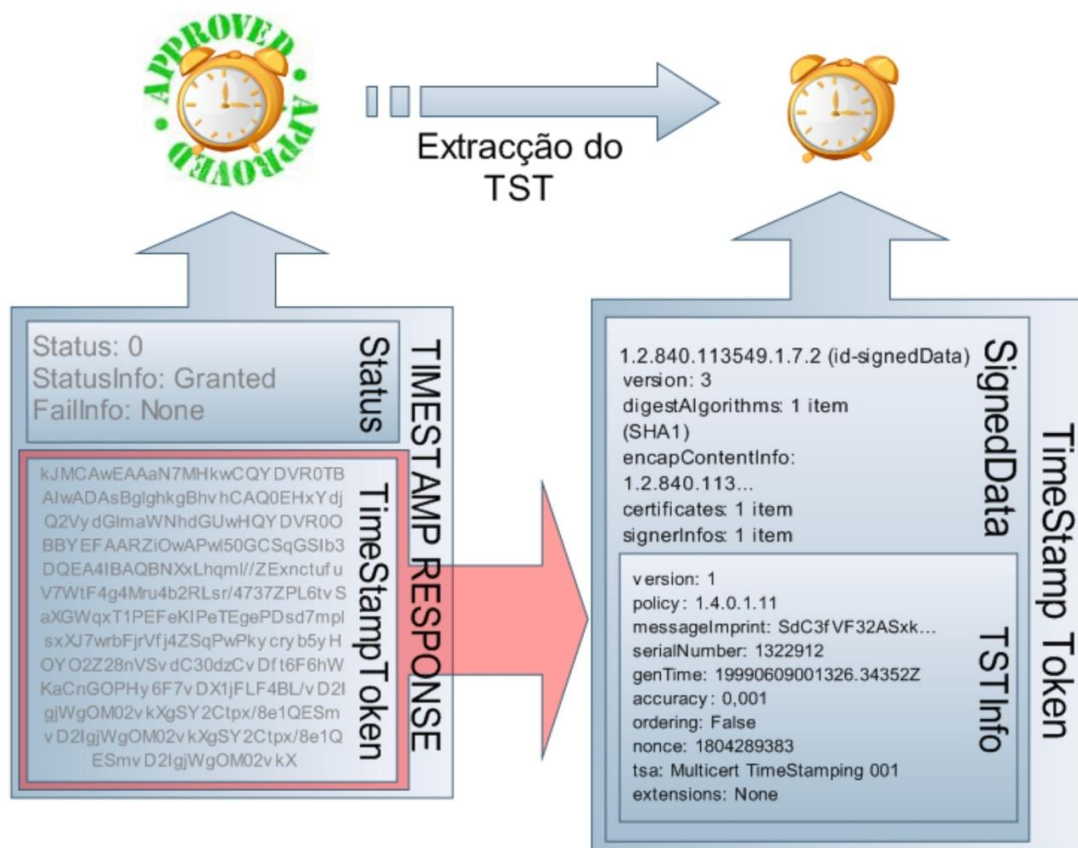


Figura 4.7 - Estrutura de um TST

Da análise do RFC 3161, pode ler-se no “anexo A”, que é aconselhado guardar um *TimeStampToken* como atributo não assinado numa estrutura CMS. Dado que o S/MIME é baseado na norma CMS, então é possível anexar o TST à estrutura, como atributo não assinado.

A solução proposta para o problema do anexo e envio do TST é implementada de acordo com o diagrama da Figura 4.8. Este contém uma mensagem assinada e um *TimeStampResponse*, sendo este último a resposta enviada da TSA. Desta resposta, só o *TimeStampToken* deve ser enviado com a mensagem. Após extrair o TST do TSR, este deverá ser anexo à assinatura.

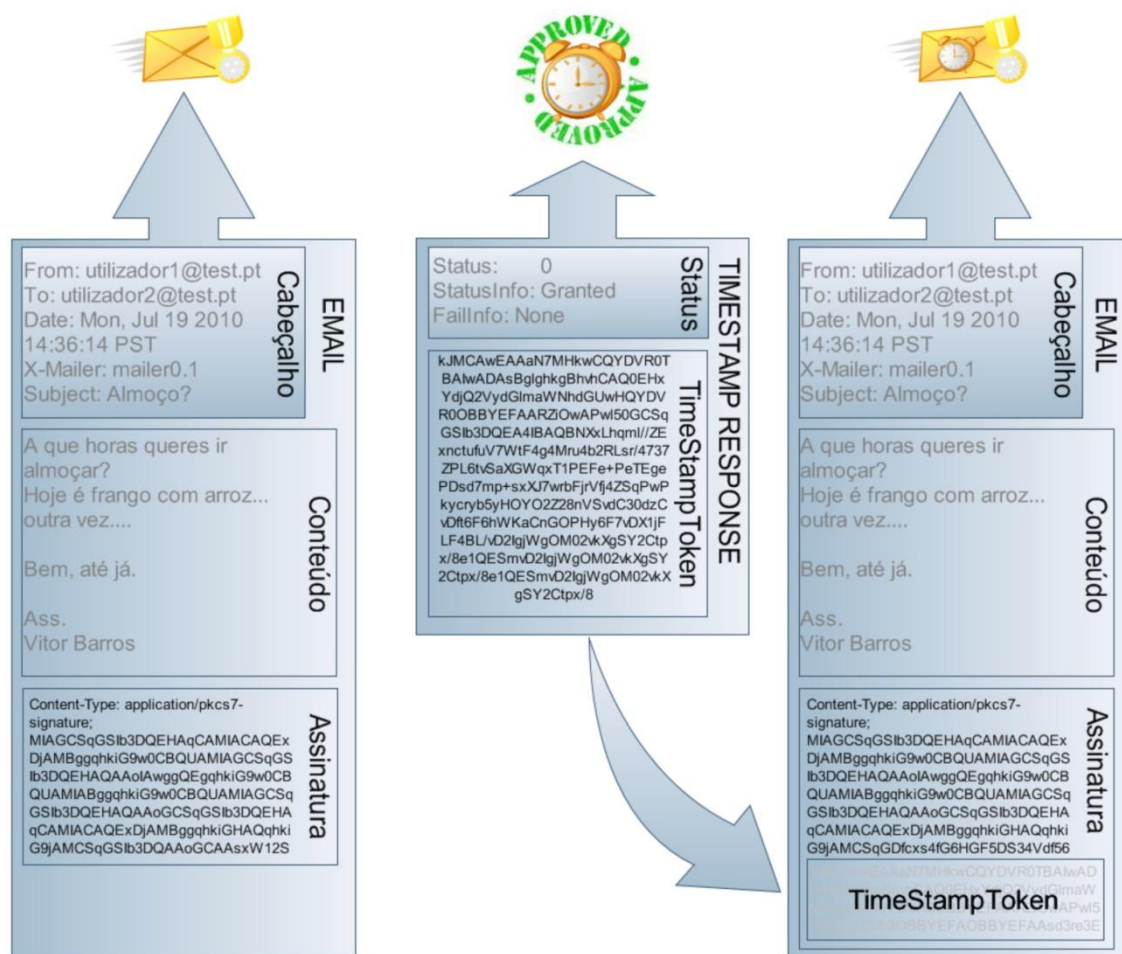


Figura 4.8 - Anexo de TST a mensagem S/MIME

Com recurso ao *timestamping* torna-se possível validar um certificado pela data emitida no *timestamp* e não a data do *Signing-Time*. Isto permite validar um certificado pela data em que o *timestamp* foi emitido, permitindo validar correctamente as mensagens cujos certificados tenham entretanto expirado ou sido revogados.

Para realizar esta implementação foi escolhido o C++ e o *OpenSSL*. O *OpenSSL* foi uma opção tomada no âmbito do projecto empresarial essencialmente devido à sua grande utilização. À data da escrita da dissertação, o *OpenSSL* já se encontra na versão 1.0.0, mas para o desenvolvimento foi usada a versão 1.0.0-beta3 (a mais recente).

Capítulo 5

Implementação

Neste capítulo é descrita a implementação da solução proposta. A implementação deu origem a dois módulos em C++ e a integração de um desses módulos no *Horde*. Na Figura 5.1 pode ver-se um diagrama ilustrativo do resultado da implementação.

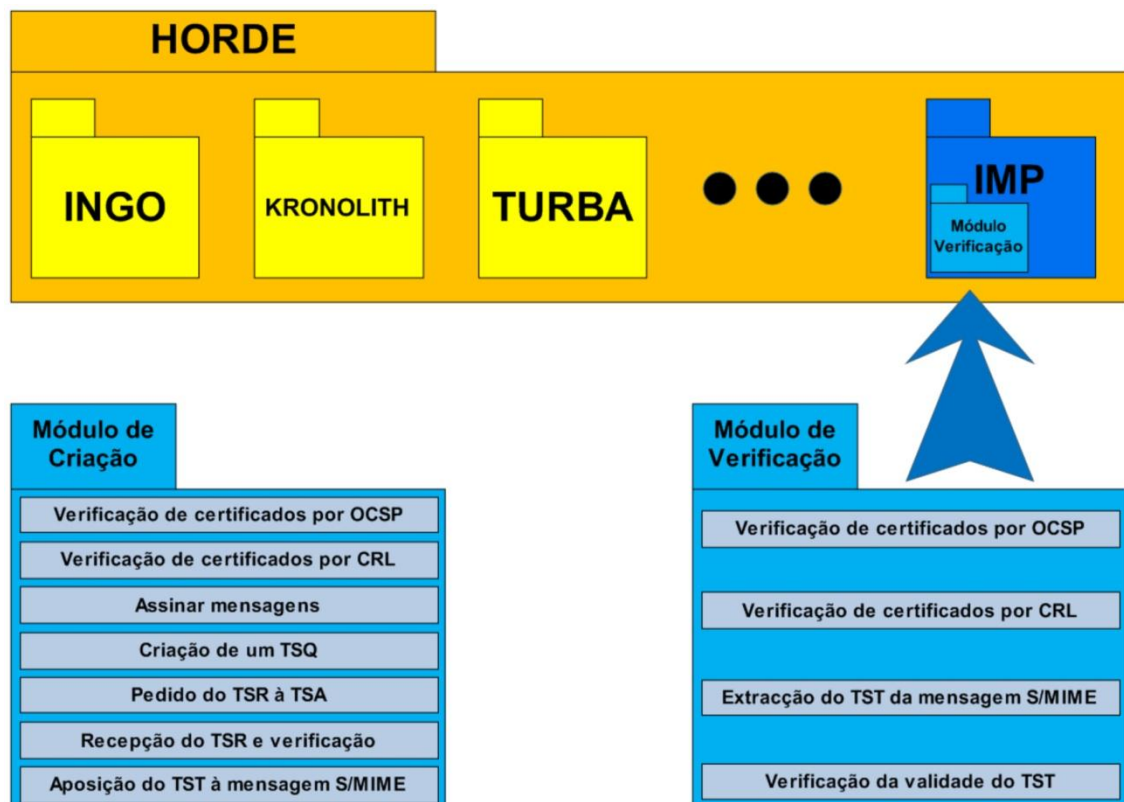


Figura 5.1 - Esquema ilustrativo da implementação

Existem dois tipos de trabalho realizado no desenvolvimento dos módulos: implementação e adaptação do código da verificação de certificados X.509 e de desenvolvimento de uma solução de *timestamps* baseada nas recomendações do RFC 3161. O código de verificação serve para validar certificados do signatário e da TSA. Para a implementação dos *timestamps* a documentação relativa é extremamente escassa. As implementações são explicadas de acordo com os módulos de trabalho realizados: módulo de criação, módulo de verificação e implementação no *Horde*.

5.1 Módulo de Criação

Nesta secção será explicada a implementação do módulo de criação (Figura 5.2). Este é composto por operações de verificações de certificados, assinatura de mensagens e processamento de *timestamps*.

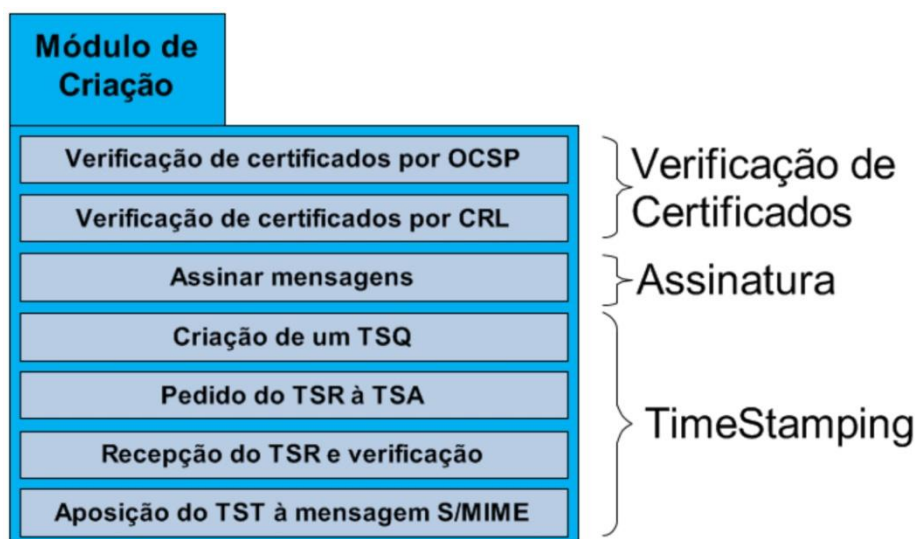


Figura 5.2 - Módulo de Criação

5.1.1 Verificação de Certificados

Devido à natureza do trabalho a ser efectuado, a verificação de certificados X.509 teve, numa fase inicial, uma especial importância. Este módulo é usado para verificar os certificados pessoais e o certificado da TSA. Na secção 5.1.1.1 e 5.1.1.2 é explicada a implementação da verificação através de OCSP e CRL, respectivamente. Na secção 5.1.1.3 é explicada a integração destes mecanismos no código final.

5.1.1.1 OCSP

O módulo de verificação por OCSP foi implementado de acordo com o diagrama de actividades apresentado na Figura 5.3. As funções implementadas no módulo recorrem ao *OpenSSL*. Esta verificação pode retornar 3 estados, certificado válido, inválido ou erro. Em caso de erro, nada se pode assumir quanto à validade do certificado pois esta não é analisada. Neste diagrama o estado de erro foi simplificado, na implementação existe um total de 12 erros diferentes que permitem depurar a origem deste.

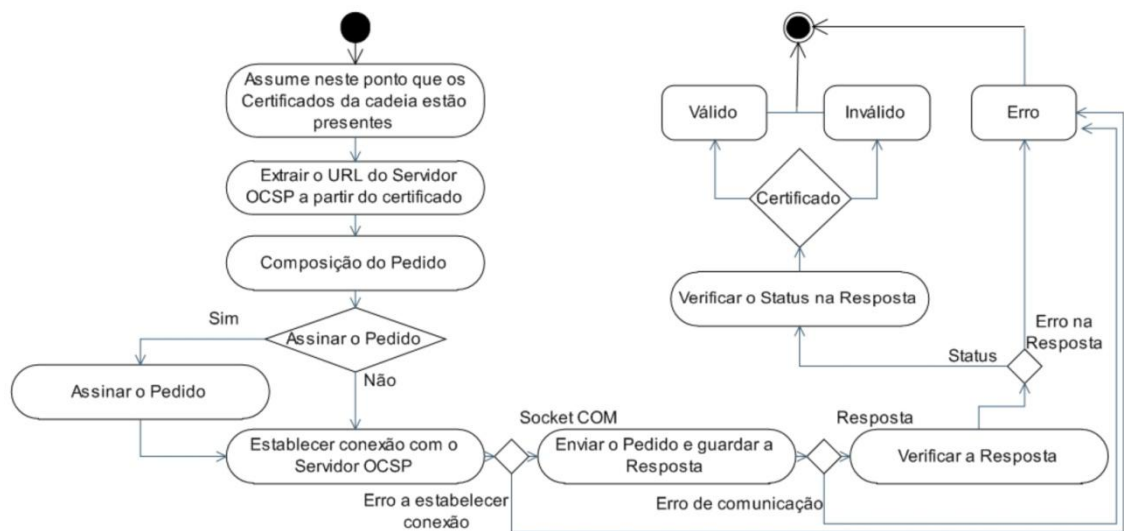


Figura 5.3 - Implementação da verificação por OCSP

Para a análise da resposta devolvida foi construído um “*switch case*” com todas as respostas possíveis do módulo de verificação OCSP.

```

switch ((int) spc res) {
case -12 :
    std::cout << "\n(Verificador OCSP) - INVALIDRESPONSE\n\n";
    goto crl;
case -11 :
    std::cout << "\n(Verificador OCSP) - CONNECTFAILURE\n\n";
    goto crl;
case -10 :
    std::cout << "\n(Verificador OCSP) - SIGNFAILURE\n\n";
    goto crl;
case -9 :
    std::cout << "\n(Verificador OCSP) - BADOCSPADDRESS\n\n";
    goto crl;
}
  
```

```
case -8 :
    std::cout << "\n(Verificador OCSP) - OUTFOFMEMORY\n\n";
    goto crl;
case -7 :
    std::cout << "\n(Verificador OCSP) - UNKNOWN\n\n";
    goto crl;
case -6 :
    std::cout << "\n(Verificador OCSP) - UNAUTHORIZED\n\n";
    goto crl;
case -5 :
    std::cout << "\n(Verificador OCSP) - SIGREQUIRED\n\n";
    goto crl;
case -3 :
    std::cout << "\n(Verificador OCSP) - TRYLATER\n\n";
    goto crl;
case -2 :
    std::cout << "\n(Verificador OCSP) - INTERNALERROR\n\n";
    goto crl;
case -1 :
    std::cout << "\n(Verificador OCSP) - MALFORMEDREQUEST\n\n";
    goto crl;
case 0:
    std::cout << "\n(Verificador OCSP) - CERTIFICATE_VALID\n\n";
    return 1;
case 1:
    std::cout << "\n(Verificador OCSP) - CERTIFICATE REVOKED\n\n";
    return 0;
}
```

Este prevê todas as respostas que o módulo implementado de OCSP possa retornar. Qualquer valor negativo é considerado erro, o 0 serve para identificar um certificado válido e o 1 para um certificado revogado. Estes valores estão de acordo com o definido no RFC 2560 [16].

5.1.1.2 CRL

O módulo de verificação por CRL foi implementado de acordo com o diagrama de actividades apresentado na Figura 5.4. Esta verificação, tal como no caso do OCSP, pode

retornar 3 estados: certificado válido, inválido ou erro. Em caso de erro nada se pode assumir quanto à validade do certificado pois esta não é analisada.

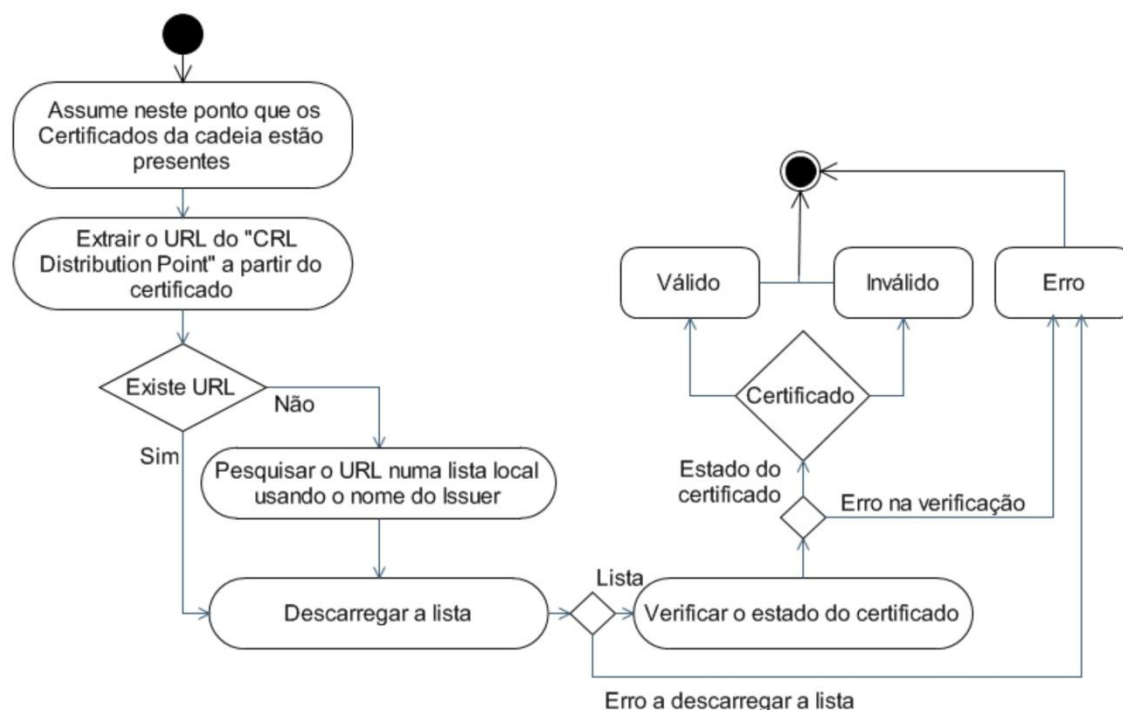


Figura 5.4 - Implementação da verificação CRL

5.1.1.3 Integração das verificações de certificados no código final

Estando disponíveis duas formas de fazer a verificação de certificados, foi tomada a decisão de dar prioridade à verificação por OCSP, só é executada uma verificação por listas CRL se a verificação por OCSP falhar. Dado que uma resposta OCSP contém menos informação que uma lista de revogação típica, é mais fácil a transmissão dos dados pela rede sem a sobrecarregar. Na implementação CRL apresentada, é sempre feito o *download* de toda a lista, este comportamento poderia ser evitado recorrendo a *caches*. Apesar de ser um comportamento previsto no código, não foi implementado devido a não apresentar muita relevância pois foi assumido que o OCSP tratará dos pedidos na maior parte dos casos, sendo a verificação por CRL um recurso. Na Figura 5.5 pode ser visto o diagrama respectivo à integração das verificações OCSP e CRL no código principal.

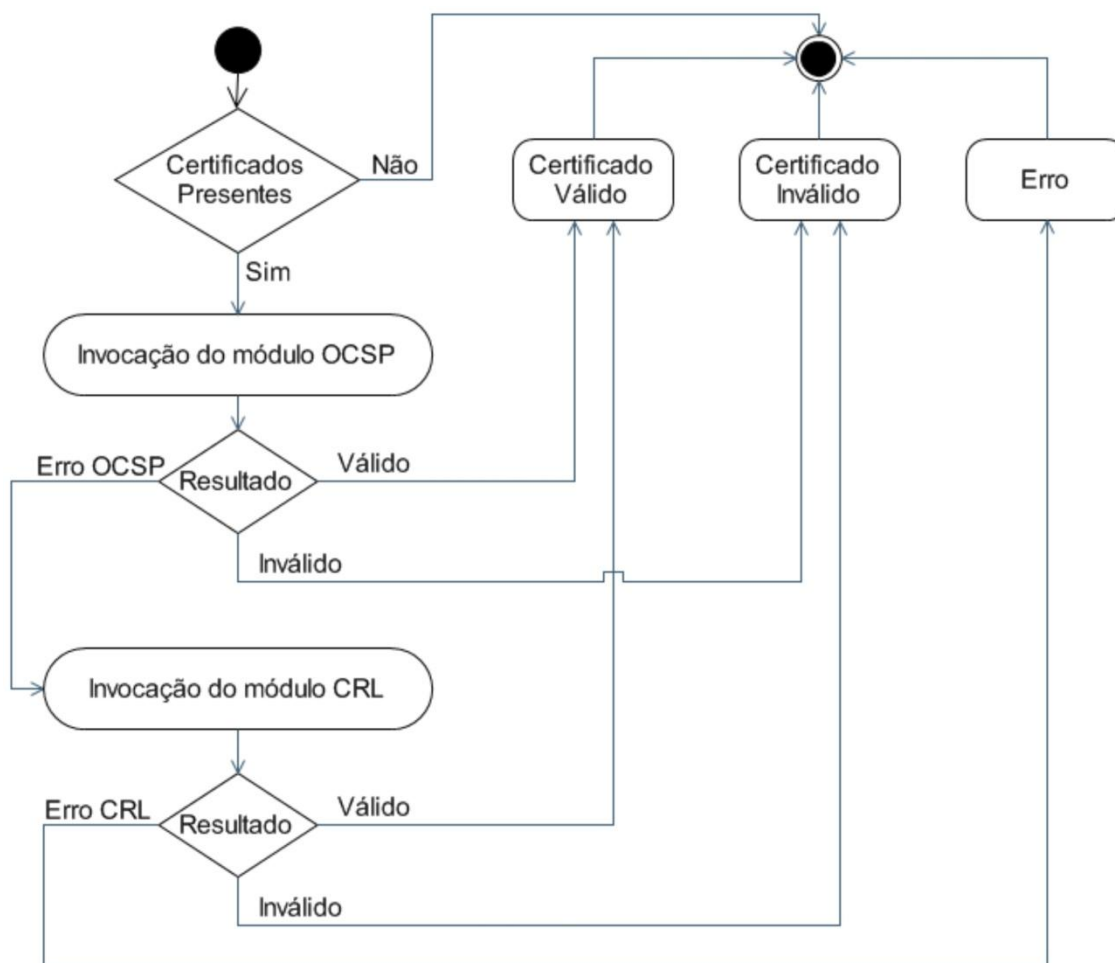


Figura 5.5 - Integração das verificações por OCSP e CRL

Explicando sucintamente a integração adoptada, em primeiro lugar, assegura-se que os certificados da cadeia estão presentes de forma a avançar com o resto das verificações. É invocado o módulo OCSP, se o resultado deste for o estado do certificado, então avança-se para o final retornando o estado válido ou inválido. Em caso de erro invoca-se o módulo de verificação por CRL, onde é mais uma vez retornado o estado do certificado (válido ou inválido). Em caso de erro no módulo CRL, avança-se para o fim com o retorno de erro. Neste caso, nada se pode assumir quanto à validade do certificado.

5.1.2 Assinatura

Após garantir a validação dos certificados, procede-se à assinatura da mensagem, esta implementação é feita recorrendo mais uma vez às ferramentas fornecidas pelo *OpenSSL*. Na Figura 5.6 pode-se ver o processo adoptado para assinar mensagens.

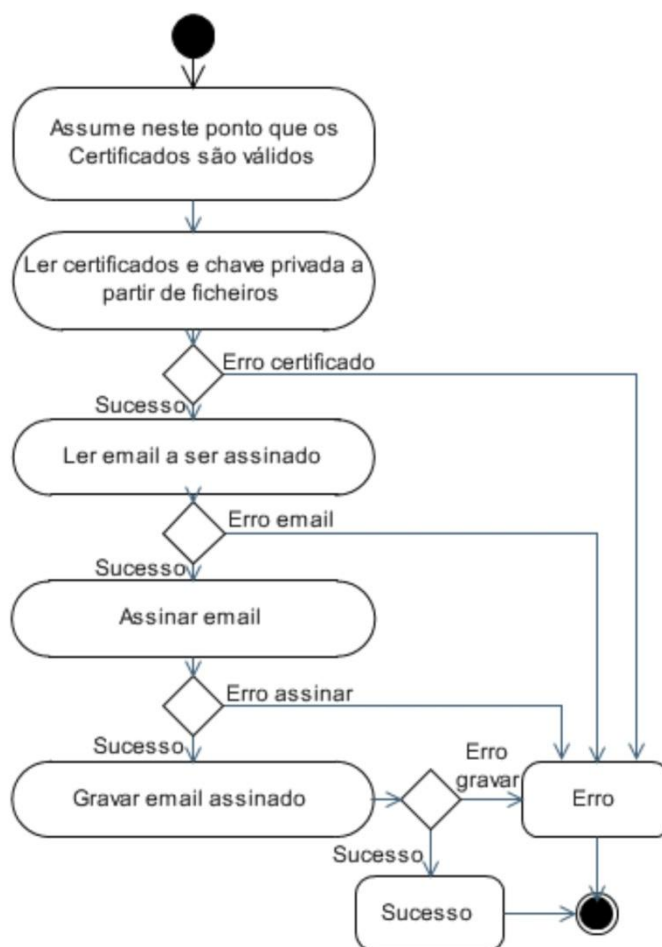


Figura 5.6 - Assinar mensagens

O processo de assinar depende de 4 funções principais do *OpenSSL* (Tabela 5.1):

Funções [Retorno Função (Argumentos)]	Descrição
X509* PEM_read_bio_X509 (<i>BIO* bp, X509* x, pem_password_cb* cb, void* u</i>)	Esta função permite ler o certificado que vai ser usado para assinar a mensagem
EVP_PKEY* PEM_read_bio_PrivateKey (<i>BIO* bp, EVP_PKEY* x, pem_password_cb* cb, void* u</i>)	Esta função permite ler a chave privada a ser usada para assinar.
PKCS7* PKCS7_sign (<i>X509 *signcert, EVP_PKEY *pkey, STACK_OF(X509) *certs, BIO *data, int flags</i>);	Cria e retorna uma estrutura PKCS7 signedData.
int SMIME_write_PKCS7 (<i>BIO *bio, PKCS7 *p7, BIO *data, int flags</i>);	Converte uma estrutura PKCS7 em uma estrutura do formato S/MIME.

Tabela 5.1 - Funções relevantes do *OpenSSL* à assinatura de mensagens

5.1.3 Timestamping

De seguida são explicadas as abordagens feitas na implementação e a sua relação com o especificado no RFC. Que se conheça não existe nenhuma implementação que permita a aplicação de *timestamps* como atributo não assinado a mensagens S/MIME, apesar de estar definida uma forma de anexá-los a estruturas CMS no RFC.

O módulo de criação aplica um *timestamp* como atributo não assinado à mensagem S/MIME. Nos pontos seguintes é descrito o processo desde a criação do *TimeStampQuery* até este ser aplicado na estrutura da mensagem S/MIME.

5.1.3.1 Criação de um TimeStampRequest³

O primeiro passo no processo de obter um *TimeStampToken* consiste na criação de um pedido. A sintaxe deste é definida no “RFC 3161 - 2.4.1. Request Format”. A estrutura de um TSQ está de acordo com a norma ASN.1 e deve ter a seguinte forma:

```
TimeStampReq ::= SEQUENCE {
    version                INTEGER { v1(1) },
    messageImprint         MessageImprint,
    reqPolicy              TSAPolicyId          OPTIONAL,
    nonce                  INTEGER              OPTIONAL,
    certReq                BOOLEAN              DEFAULT FALSE,
    extensions              [0] IMPLICIT Extensions OPTIONAL }
```

O *messageImprint* é uma estrutura definida da seguinte forma:

```
MessageImprint ::= SEQUENCE {
    hashAlgorithm          AlgorithmIdentifier,
    hashedMessage           OCTET STRING }
```

O *OpenSSL* contém a definição de uma variável chamada *TS_REQ*, esta é uma variável do tipo “*TimeStampReq*”. Existem funções específicas para interagir com esta variável definidas no *OpenSSL*. De forma a construir o TSQ foi criada uma classe que permite, de uma forma mais simples, interagir com uma variável *TS_REQ*. A classe chamada *TimeStampRequest* funciona como um *wrapper* das funções fornecidas no código do *OpenSSL*. Esta possui funções do tipo *GETTERS* (Tabela 5.2), funções de leitura, e *SETTERS* (Tabela 5.3), funções de escrita que permitem preencher o TSQ:

³ Dado que *TimeStampRequest* e *TimeStampResponse* dão ambos origem à sigla TSR, neste documento, TSQ (*TimeStampQuery*) refere-se ao *TimeStampRequest* e TSR (*TimeStampReply*) refere-se ao *TimeStampResponse*. Esta nomenclatura está de acordo com a usada no RFC 3161.

GETTERS	Descrição
	Função OpenSSL usada internamente
int getVersion();	Retorna a versão do TSQ <i>long TS_REQ_get_version(const TS_REQ *a)</i>
AlgoID getDigestAlgorithm();	Retorna o Identificador do Algoritmo usado no cálculo da Hash (pertence ao MessageImprint) <i>X509_ALGOR</i> <i>*TS_MSG_IMPRINT_get_algo(TS_MSG_IMPRINT *a);</i>
std::string getHashedMessage();	Retorna a Hashed Message (pertence ao MessageImprint) <i>TS_MSG_IMPRINT *TS_REQ_get_msg_imprint(TS_REQ *a);</i>
OID getPolicyID();	Retorna o ObjectIdentifier usado para definir a reqPolicy <i>ASN1_OBJECT *TS_REQ_get_policy_id(TS_REQ *a);</i>
long int getNonce();	Retorna o Nonce <i>const ASN1_INTEGER *TS_REQ_get_nonce(const TS_REQ *a);</i>
bool getCertificateRequest();	Retorna o estado do certReq <i>int TS_REQ_get_cert_req(const TS_REQ *a);</i>

Tabela 5.2 - GETTERS da classe TimeStampRequest

SETTERS [Retorno Função (Argumentos)]	Descrição
	Função OpenSSL usada internamente
bool setVersion (long int version);	Preenche a Versão no TSQ int TS_REQ_set_version(TS_REQ *a, long version);
bool setDigestAlgorithm (AlgoID algorithm);	Esta função invoca uma outra classe chamada MessageImprint. É apresentada a função final da classe MessageImprint aqui int TS_MSG_IMPRINT_set_algo(TS_MSG_IMPRINT *a, X509_ALGOR *alg);
bool setHashedMessage (const std::string hash);	Esta função invoca uma outra classe chamada MessageImprint. É apresentada a função final da classe MessageImprint aqui int TS_MSG_IMPRINT_set_msg(TS_MSG_IMPRINT *a, unsigned char *d, int len);
bool setMessageImprint (AlgoID algorithm, const std::string data);	Preenche o MessageImprint (executa as duas funções anteriores com uma invocação) no TSQ int TS_MSG_IMPRINT_set_algo(TS_MSG_IMPRINT *a, X509_ALGOR *alg); int TS_MSG_IMPRINT_set_msg(TS_MSG_IMPRINT *a, unsigned char *d, int len);
bool setPolicyID (OID policy);	Preenche o campo reqPolicy no TSQ int TS_REQ_set_policy_id(TS_REQ *a, ASN1_OBJECT *policy);
bool setNonce (long int nonce);	Preenche o campo Nonce no TSQ int TS_REQ_set_nonce(TS_REQ *a, const ASN1_INTEGER *nonce);
bool setCertificateRequest (bool cert_req);	Preenche o campo certReq no TSQ int TS_REQ_set_cert_req(TS_REQ *a, int cert_req);

Tabela 5.3 - SETTERS da classe TimeStampRequest

As funções da classe possuem um tratamento prévio e posterior dos dados, sendo a invocação da função *OpenSSL* uma parte da implementação. Fica a lista das funções para melhor se entender a interacção com o *OpenSSL*. A classe possui mais funções que as definidas nas tabelas mas para a construção do TSQ, não são relevantes.

Dispondo então de todas estas funções, o primeiro passo para criar o TSQ é calcular a *hash* da mensagem de email da qual queremos pedir o TST. Foi usada a seguinte função para esse propósito:

```
SHA1((const unsigned char*) mail_in.c_str(), mail_in.size(),
sha1_hash);
```

Preenche-se de seguida o TSQ invocando as funções *SETTERS*:

```
tsq.setCertificateRequest(true);
tsq.setDigestAlgorithm(AlgoID(OID::SHA1()));
tsq.setHashedMessage(std::string((char*) sha1_hash, 20));
tsq.setVersion(1);
tsq.setNonce(random());
OID::define("1.4.0.1.11", "TSA Policy", "Multicert TSA Policy");
tsq.setPolicyID(OID(OID::oid2NID("1.4.0.1.11")));
```

É necessário um correcto preenchimento da política usada de forma a este pedido ser correctamente processado pela TSA. Portanto, houve necessidade de tornar este pedido de acordo com o esperado pela TSA da MultiCert, empresa que disponibilizou um TSA de teste para emissão dos *timestamps* neste projecto. Temos agora o TSQ devidamente construído faltando somente enviar para a TSA.

5.1.3.2 Pedido do *TimeStampResponse* à TSA

Para efectuar o pedido à TSA é necessário enviar o TSQ para o servidor via HTTP. É construído um pacote com o cabeçalho e dados, este é enviado para a TSA recorrendo à função *sendToTSA* definida na classe *TimeStampRequest*, esta função retorna a resposta em bruto enviada pela TSA:

```
std::string TimeStampRequest::sendToTSA(const std::string host,
const std::string relative_path){

    std::string data = der_data();

    std::string str = "POST " + relative_path + " HTTP/1.0\r\n";
    str += "User-Agent: Simple Client v0.01\r\n";
    str += "Host: " + host + "\r\n";
```

```

str += "Content-Type: application/timestamp-query\r\n";
str += "Accept: application/timestamp-reply\r\n";
str += "Content-Length: " + int2String(data.size()) + "\r\n\r\n";
str += data;
(. . .)
return std::string(response, len);

```

Do código anterior é de realçar que a função *der_data()* está definida na classe *TimeStampRequest*, esta retorna os dados no formato de codificação *Distinguished Encoding Rules* (DER) [4]. Do cabeçalho, os campos “Content-type:” e “Accept:” estão definidos no RFC 3161 - 3.4. *Time-Stamp Protocol* via HTTP. Todos os outros campos são normas do protocolo de transporte.

5.1.3.3 Verificação do TimeStampResponse recebido

Dado que a resposta foi recebida via HTTP, é necessário retirar a parte correspondente ao *timestamp*:

```

int garbage = ts.find("\r\n\r\n");

if (garbage == std::string::npos) {
    std::cout << "Well..." << std::endl;
    return -1;
}

ts = ts.substr(garbage + 4);

```

De seguida vai ser explicada a estrutura de um TSR de forma a melhor se entender os passos tomados a seguir. A sintaxe deste é definida no “RFC 3161 - 2.4.2. *Response Format*”.

```

TimeStampResp ::= SEQUENCE {
    status                PKIStatusInfo,
    timeStampToken        TimeStampToken    OPTIONAL }

```

O *status* é baseado na definição de *status* da secção 3.2.3 do [RFC 2510] [17]:

```

PKIStatusInfo ::= SEQUENCE {
    status          PKIStatus,
    statusString    PKIFreeText    OPTIONAL,
    failInfo        PKIFailureInfo OPTIONAL }

```

Quando o *status* contém o valor zero ou um deve estar presente um *TimeStampToken*, para os outros valores não deve estar presente um *TimeStampToken*. Um dos valores seguintes deve estar contido no *status*:

```

PKIStatus ::= INTEGER {
    granted              (0),
    grantedWithMods      (1),
    rejection            (2),
    waiting              (3),
    revocationWarning    (4),
    revocationNotification (5)
}

...

```

O *TimeStampToken* é definido como se segue.

```

TimeStampToken ::= ContentInfo{
    (...)
    id-ct-TSTInfo OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) ct(1) 4}
}

TSTInfo ::= SEQUENCE {
    version                INTEGER { v1(1) },
    policy                 TSAPolicyId,
    messageImprint         MessageImprint,
    serialNumber           INTEGER,
    genTime                GeneralizedTime,
    accuracy               Accuracy OPTIONAL,
    ordering               BOOLEAN    DEFAULT FALSE,
    nonce                 INTEGER    OPTIONAL,
    tsa                   [0] GeneralName OPTIONAL,

```

extensions	[1] IMPLICIT Extensions	OPTIONAL
}		

É necessário proceder à verificação da validade do TSR. Para melhor compreender os campos a analisar, com base RFC 3161 - 2.2 TSA Transactions retira-se o mecanismo de validação com os seguintes pontos:

Após receber a resposta a entidade que executa o pedido deve **verificar o erro de *status*** retornado na resposta e se não encontrar nenhum erro, deve **verificar os vários campos contidos no *TimeStampToken* e a validade da assinatura digital do *TimeStampToken***. Em particular deve **verificar que o *messageImprint* corresponde ao que foi pedido para assinar com o *TimeStampRequest***. A entidade que executou o pedido deve **verificar que o *TimeStampToken* contém o certificado correcto do TSA, a data correcta e o algoritmo *hash* OID correcto**. Deve também **verificar o valor do *nonce*** (um número grande e aleatório com alta probabilidade de ser gerado pelo cliente apenas uma vez) **incluído na resposta contra o valor incluído no *TimeStampRequest***. Se **qualquer das verificações anteriores falhar, o *TimeStampToken* deve ser rejeitado**.

De seguida são explicados os passos tomados para obedecer a estes parâmetros de verificação. Inicialmente houve a necessidade de colocar a resposta em estruturas que pudessem ser usadas no código e compreendidas pelas funções do *OpenSSL*:

```
MemoryBIO tmp;
tmp.write(ts);

TS_RESP *response = NULL;

std::cout << "MemoryBIO has " << tmp.size() << std::endl;

std::string ts_resp;
tmp.peek(ts_resp);

response = d2i_TS_RESP_bio(tmp.getBIO(), NULL);

(. . .)

if (!response) {
    std::cout << "Error deserializing TimeStampResponse!" <<
std::endl;
```

```
        return -1;
    }

    std::cout << "TS_RESP sucessfully obtained! (" << (int)
response << ")" << std::endl;
    int size = i2d_PKCS7(response->token, &pbuf);
    std::cout << "TOKEN SIZE: " << size << std::endl;

    (. . .)

    std::string ci((char*)buf, size);
    TimestampToken tst(ci);
```

Existem duas variáveis importantes aqui criadas, a “**response**” contém o **TSR**, e a “**tst**” contém o **TST_info**. A classe *TimestampToken* foi criada, tal como a *TimestampRequest*, de forma a funcionar como um wrapper das funções do OpenSSL. Internamente a classe utiliza uma variável do tipo **TS_TST_INFO** definida no *OpenSSL*. Na Tabela 5.4 estão definidas as funções relevantes presentes nesta classe. Estas funções permitem interagir com o **TSTinfo**, apesar de para a verificação da validade não ser suficiente, foi introduzido aqui pois é durante a verificação que aparece a necessidade de ler os campos que foram preenchidos e enviados pela TSA.

Funções da classe [Retorno Função (<i>Argumentos</i>)]	Descrição
	Função OpenSSL usada internamente
TimeStampToken (std::string data)	Construtor da classe, cria a variável usando uma <i>string</i> de dados Utiliza várias funções do OpenSSL
long int version ()	Retorna a versão do TSTInfo long TS_TST_INFO_get_version(const TS_TST_INFO *a);
OID tsaPolicyId ()	Retorna a <i>PolicyId</i> da TSA ASN1_OBJECT *TS_TST_INFO_get_policy_id(TS_TST_INFO *a);
AlgID hashAlgorithm ()	Retorna o algoritmo usado no cálculo da <i>Hash</i> X509_ALGOR *TS_MSG_IMPRINT_get_algo(TS_MSG_IMPRINT *a);
std::string hashedMessage ()	Retorna a <i>Hash</i> na mensagem ASN1_OCTET_STRING *TS_MSG_IMPRINT_get_msg(TS_MSG_IMPRINT *a)
long int serialNumber ()	const ASN1_INTEGER *TS_TST_INFO_get_serial(const TS_TST_INFO *a);
std::string time ()	Retorna a Data, em formato legível, da criação do <i>TimeStampToken</i> . const ASN1_GENERALIZEDTIME *TS_TST_INFO_get_time(const TS_TST_INFO *a);
std::string accuracy ()	Retorna a precisão usada na data TS_ACCURACY *TS_TST_INFO_get_accuracy(TS_TST_INFO *a);
bool ordering ()	Retorna o estado do campo <i>ordering</i> int TS_TST_INFO_get_ordering(const TS_TST_INFO *a);
long int nonce ()	Retorna o <i>Nonce</i> const ASN1_INTEGER *TS_TST_INFO_get_nonce(const TS_TST_INFO *a);

Tabela 5.4 - Funções da classe *TimeStampToken*

Para proceder à verificação da resposta, é necessário possuir o TSR e o TSQ por motivos de comparação de dados (nomeadamente o *Nonce* e o *Hash* da mensagem). A verificação, após serem criadas e preenchidas as variáveis a ser usadas, é feita invocando a seguinte função:

```
int TS_RESP_verify_response(TS_VERIFY_CTX *ctx, TS_RESP *response);
```

A invocação desta função executa as seguintes verificações:

TS_RESP_verify_response() verifica se o TSR contém uma resposta válida no que respeita a definições de contexto:

- Dá uma mensagem de erro se o TS_TST_INFO não estiver presente.
- Invoca TS_RESP_verify_token() para verificar o conteúdo do token.

É invocada a função:

```
int TS_RESP_verify_token(TS_VERIFY_CTX *ctx, PKCS7 *token);
```

A invocação desta função executa as seguintes verificações:

Verifica se o token "contém um *TimeStampToken* válido" no que diz respeito às definições do contexto.

Apenas as verificações especificadas abaixo são executadas:

- Verifica a assinatura do TS_TST_INFO
- Verifica o número da versão da resposta
- Verifica se as políticas pedidas e retornadas batem certo
- Verifica se as *message imprints* são as mesmas.
- Verifica se os nonces são iguais
- Verifica se o nome da TSA bate certo com o da assinatura
- Verifica se o nome da TSA é a TSA esperada.

A maior parte dos dados estão neste ponto verificados, ficando a faltar somente a verificação da assinatura da TSA. É invocada a função:

```
int TS_RESP_verify_signature(PKCS7 *token, STACK_OF(X509) *certs,  
X509_STORE *store, X509 **signer_out);
```

Assumindo que as funções retornaram um estado de sucesso nas verificações, é possível passar para a aposição do TST à mensagem S/MIME.

5.1.3.4 Aposição do TimeStampToken ao email

Para anexar o TST à mensagem S/MIME é necessário conhecer uma forma de anexar este à mensagem. Recorrendo ao “RFC 3161 - APPENDIX A - Signature Time-stamp attribute using CMS”:

(. . .)

Um lugar aconselhável para armazenar um TST é numa estrutura CMS como um atributo não assinado.

(. . .)

O identificador do objecto a seguir identifica a assinatura do atributo Time-stamp:

id-aa-timeStampToken OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) aa(2) 14 }

(. . .)

O valor do atributo da assinatura do *TimeStampToken* tem o tipo ASN.1 *SignatureTimeStampToken*:

SignatureTimeStampToken ::= *TimeStampToken*

Deve-se anexar este TST como atributo não assinado a uma estrutura CMS. O mecanismo adoptado acaba por ser parecido com o de assinar mensagens, mas antes de devolver a mensagem é adicionado o TST. Na Figura 5.7 um diagrama de actividades mostra como é adicionado este passo à implementação.

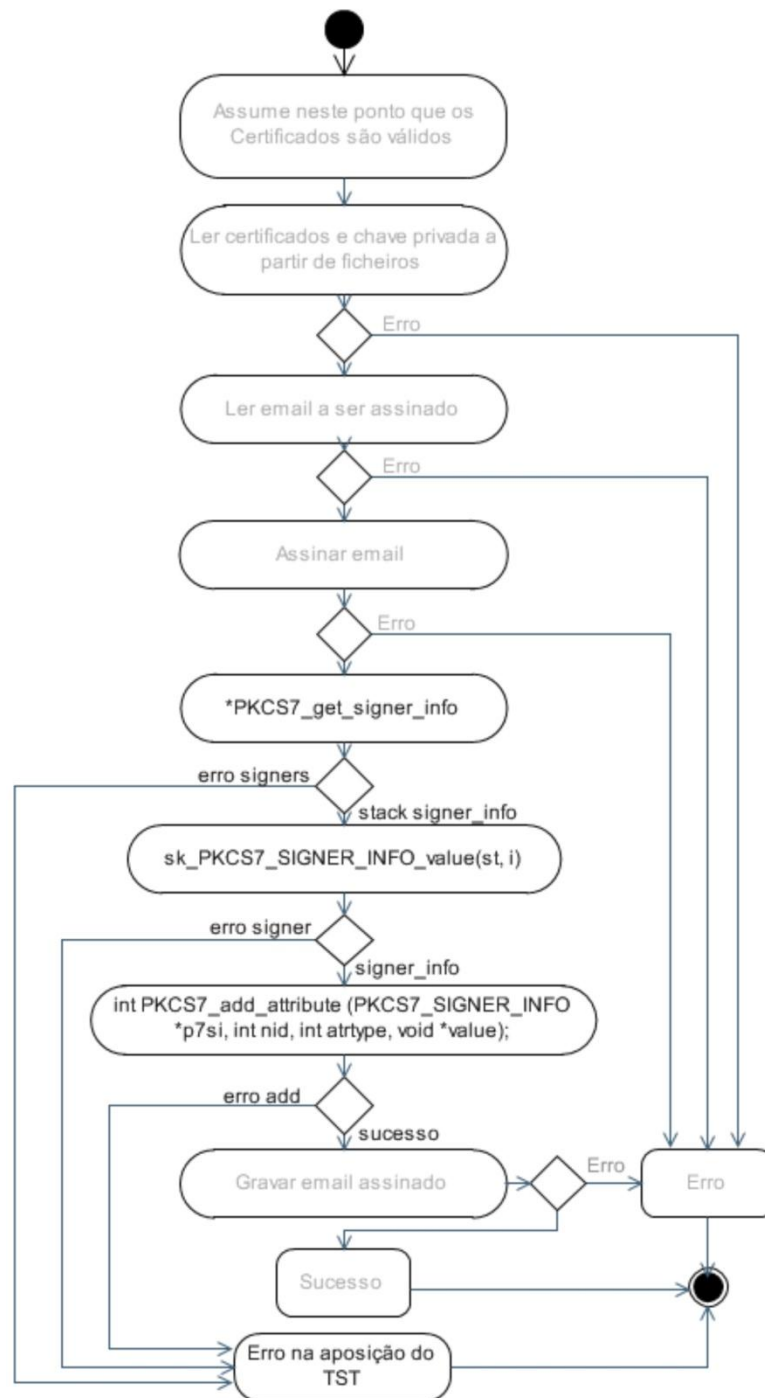


Figura 5.7 - Anexar o TST à mensagem S/MIME

Nesta figura estão exemplificadas as 3 funções principais usadas no processo. A primeira função (*PKCS7_get_signer_info*) retorna todos os *SignerInfos* presentes na estrutura PKCS7 para uma variável do tipo *stack* que vai conter todas as assinaturas presentes. A segunda função (*sk_PKCS7_SIGNER_INFO_value*) extrai dessa *stack* o *SignerInfo* ao qual queremos adicionar o TST como atributo não assinado. A terceira função necessita uma explicação um pouco mais alongada:

```
PKCS7_add_attribute(sinfo1, NID_id_smime_aa_timeStampToken,  
V_ASN1_SEQUENCE, seq);
```

No *OpenSSL* o NID funciona como um identificador numérico associado ao OID. A definição interna do *OpenSSL* associa ao *NID_id_smime_aa_timeStampToken* o OID com o valor “1.2.840.113549.1.9.16.2.14”. Este é o valor especificado no RFC 3161. O tipo de dados do atributo é definido como *ASN1_SEQUENCE*, modo definido no RFC 3161 para esta estrutura. O processo de *timestamping* no módulo de criação está concluído, a mensagem S/MIME possui agora um TST que prova a existência da mensagem à data do pedido de *timestamping*.

5.2 Módulo de Verificação

Nesta secção será explicada a implementação do módulo de verificação (Figura 5.8). Este é composto por operações de verificações de certificado e processamento de *timestamps*.

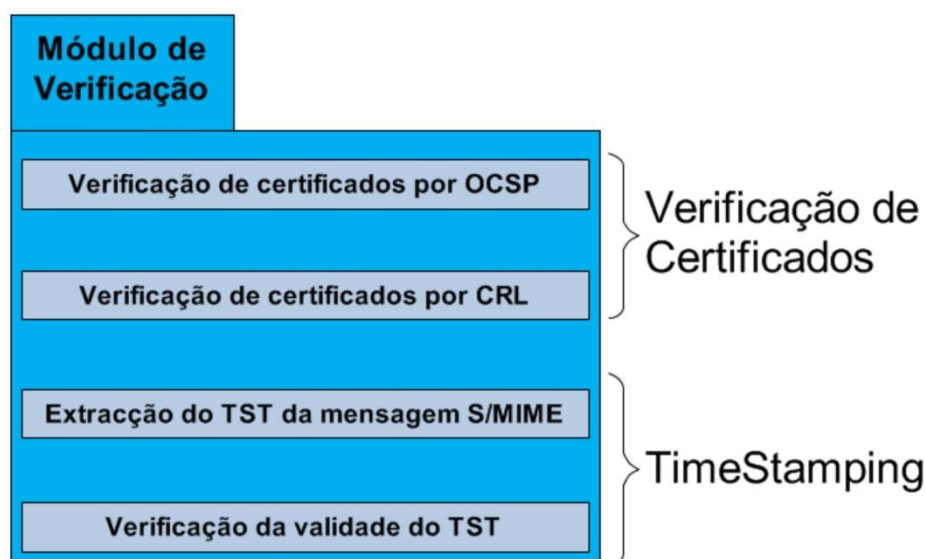


Figura 5.8 - Módulo de Verificação

5.2.1 Verificação de Certificados

A verificação de certificados foi implementada da mesma forma que no módulo de criação (5.1.1).

5.2.2 *Timestamping*

O módulo de verificação é responsável pela validação de um TST aplicado a uma mensagem S/MIME. Este foi integrado no *Horde* permitindo a verificação de um TST numa mensagem recebida. De forma a poder ser invocado como função do sistema, o programa aceita como argumento de entrada uma estrutura S/MIME e devolve os dados relativos ao TST após validação.

5.2.2.1 Extracção do TimeStampToken da mensagem S/MIME

Dado que este módulo foi implementado no *Horde*, sofreu algumas alterações de forma a poder ser integrado. Na Figura 5.9 pode ver-se o mecanismo adoptado para extrair o TST da mensagem. Trata-se de um método recursivo que pesquisa para cada assinatura presente na estrutura de dados os atributos não assinados desta, e para cada um destes atributos não assinados compara o OID com “1.2.840.113549.1.9.16.2.14” de forma a descobrir se esse atributo é um TST. No caso de ser encontrado um TST procede-se à verificação da validade deste.

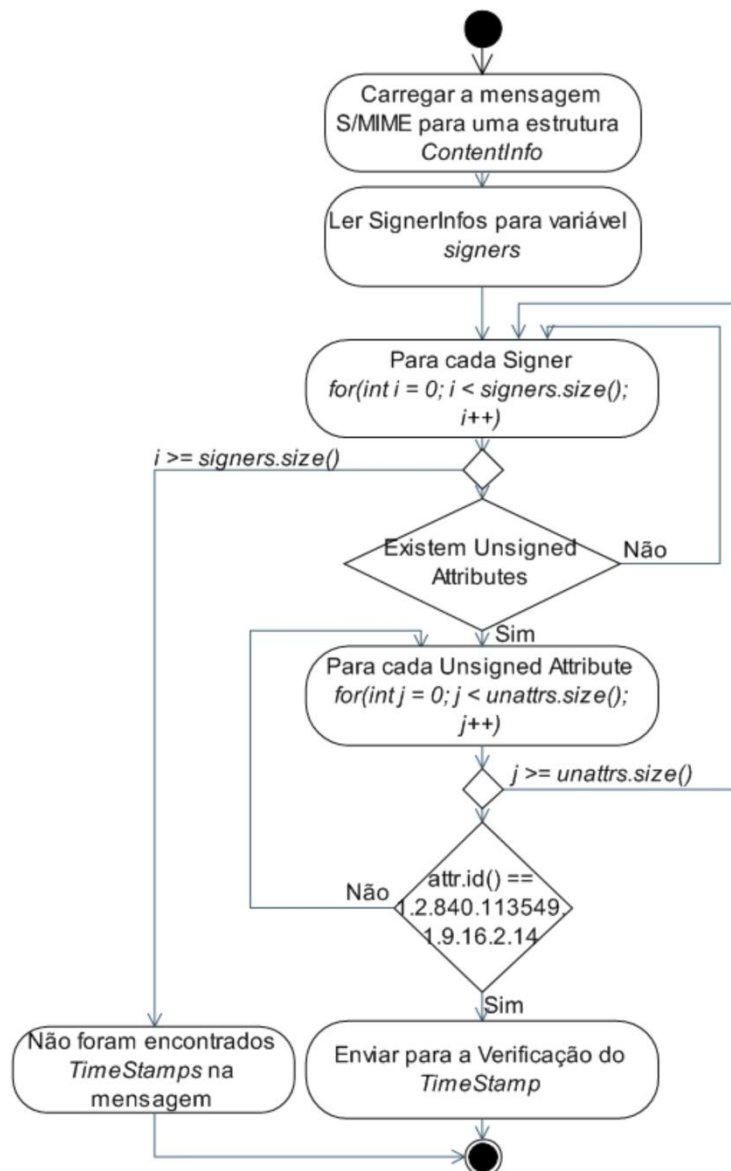


Figura 5.9 - Método de extracção de um TST a partir de um email

5.2.2.2 Verificação da validade do TimeStampToken

A validação do TST no módulo de verificação implica a validação do certificado da TSA e a comparação da *hash* contida no TST com uma *hash* calculada localmente, se o certificado retornar um estado válido e a *hash* for igual, então pode assumir-se que o TST está válido e não foi alterado. A implementação da validação de certificados é explicada na secção 5.1.1.

É necessário de seguida executar uma comparação de *hashs*. Para o cálculo da *hash* foi usada a função:

```
SHA1((const unsigned char*) mail.getData(), mail.dataSize(),
sha1_hash);
```

Na Figura 5.10 pode ver-se como foi integrado o sistema de verificação.

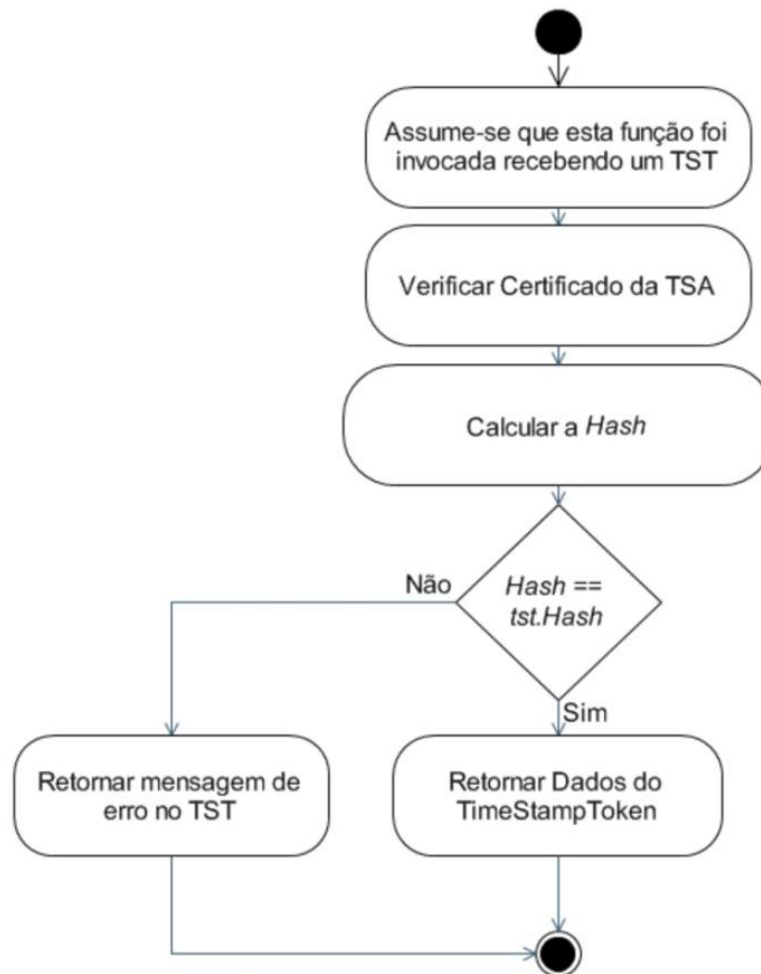


Figura 5.10 - Verificação do TST no módulo de Verificação

5.3 Integração no *Horde* do módulo de verificação

A implementação no *Horde* foi realizada no módulo IMP, o módulo da GUI de processamento de mensagens (Figura 5.11).



Figura 5.11 - Integração Horde

Para a implementação no *Horde* adicionou-se o módulo de verificação como função do sistema. O sistema operativo usado na aplicação do *Horde* foi o *CentOS*, para adicionar o módulo foi necessário compilar o programa e colocá-lo no seguinte destino:

```
/usr/bin/smimevalidator
```

Foi necessário instalar o *OpenSSL* 1.0.0-beta3 no *CentOS* pois o programa não é “autónomo” e depende de várias bibliotecas partilhadas. Após isto passou a ser possível usar o módulo desenvolvido como uma invocação de função do sistema, o que para a implementação no PHP provou ser uma alternativa bastante acessível. A nível do *Horde*, houve necessidade de fazer algumas alterações ao IMP de forma a integrar o módulo e poder invocar a função *smimevalidator*.

A pasta onde o IMP pode ser encontrado no sistema é:

```
/var/www/html/horde/imp/
```

Após a análise do código PHP do *Horde*, o ficheiro “*view.php*” apareceu como alternativa para a implementação da verificação TST. A invocação do módulo de verificação no PHP foi feita de acordo com o diagrama da Figura 5.12. Para carregar a mensagem foi usado o comando:

```
$msg = $contents->fullMessageText();
```

A preparação dos dados passou simplesmente por colocar o comando numa variável.

```
$command = "smimevalidator";
```

A invocação, apesar de simples, merece uma especial atenção devido à forma como o argumento é passado:

```
$result = system("$command '$msg'');
```

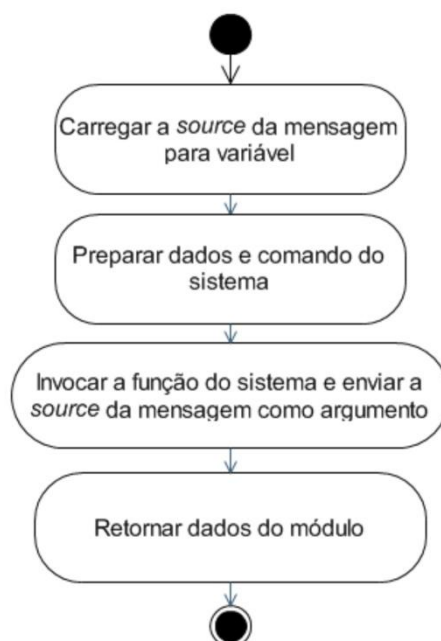


Figura 5.12 - Implementação do módulo de verificação no `view.php`

Um programa pode receber múltiplos argumentos, a forma de os separar é através do carácter espaço. Como pretendemos enviar a mensagem completa, esta contém múltiplos espaços, o que enviaria a mensagem como vários argumentos em vez de um único. A forma de resolver este problema é enviar o argumento dentro de plicas, desta forma a mensagem é recebida como um único argumento.

A apresentação da informação do *timestamp* é apresentada ao utilizador recorrendo a uma janela “*pop-up*” que apresenta vários dados como a data de criação do TST se este estiver válido, senão é apresentado um erro informando sobre a validade do TST.

5.4 Discussão

Os dois módulos foram implementados de acordo com o previsto. No entanto, para a implementação dos *timestamps* a documentação relativa é extremamente escassa. O *OpenSSL* só introduziu o código de *timestamp* na versão 1.0.0beta1 sendo a documentação de suporte reduzida, pelo que a implementação deste provou-se particularmente complicada. Devido a esta “complicação”, houve necessidade de fazer um estudo das ferramentas fornecidas pelo *OpenSSL*. Este estudo passou pela análise dos *Headers* e código de implementação.

Capítulo 6

Demonstração e Validação

Neste capítulo são focados os aspectos dos testes efectuados e os resultados obtidos. De forma geral, as demonstrações são apresentadas na forma de capturas de imagem. As demonstrações incidem sobre 3 grandes secções: módulo de criação, módulo de verificação e implementação no *Horde*.

6.1 Módulo de Criação

O módulo de criação executa operações de validação de certificados, assinatura de mensagens e aplicação de *timestamps em S/MIME*. A validação destes será feita nesta ordem.

6.1.1 Verificação de Certificados

A verificação de certificados depende, essencialmente, da verificação OCSP e CRL. Aqui são apresentadas as demonstrações relativas a dois passos intermédios, cuja validação implica o correcto funcionamento de todas as estruturas intermédias, evitando-se assim uma validação campo a campo. Inicialmente é apresentada uma demonstração da comunicação de rede destes protocolos, validando assim todos os campos de criação de pedidos e comunicação HTTP. Posteriormente é demonstrado resultado final retornado pelo *OpenSSL* implicando assim o correcto processamento das respostas. Relativamente às verificações de certificados no módulo de criação foi feita uma simplificação para testes: os certificados usados na verificação e na assinatura não são iguais. A razão desta simplificação deve-se ao facto de não se possuir certificados com chave privada que pudessem ser usados na verificação OCSP. Foi usada a cadeia de certificados da FEUP para a verificação de

certificados e posteriormente, para assinar, foi usado um certificado emitido localmente por uma CA criada para este propósito.

6.1.1.1 OCSP

De forma a validar a comunicação de rede, foi usado o *Wireshark*. Executou-se o módulo de criação de mensagens, mantendo o *Wireshark* a capturar o tráfego de rede durante o tempo de processamento do programa. Após o término deste são analisados os pacotes capturados. Na Figura 6.1 está a captura relativa ao pedido OCSP e na Figura 6.2 está a captura relativa à resposta OCSP recebida. O pedido contém os campos que identificam o certificado para o qual se pretende receber o estado da validade.

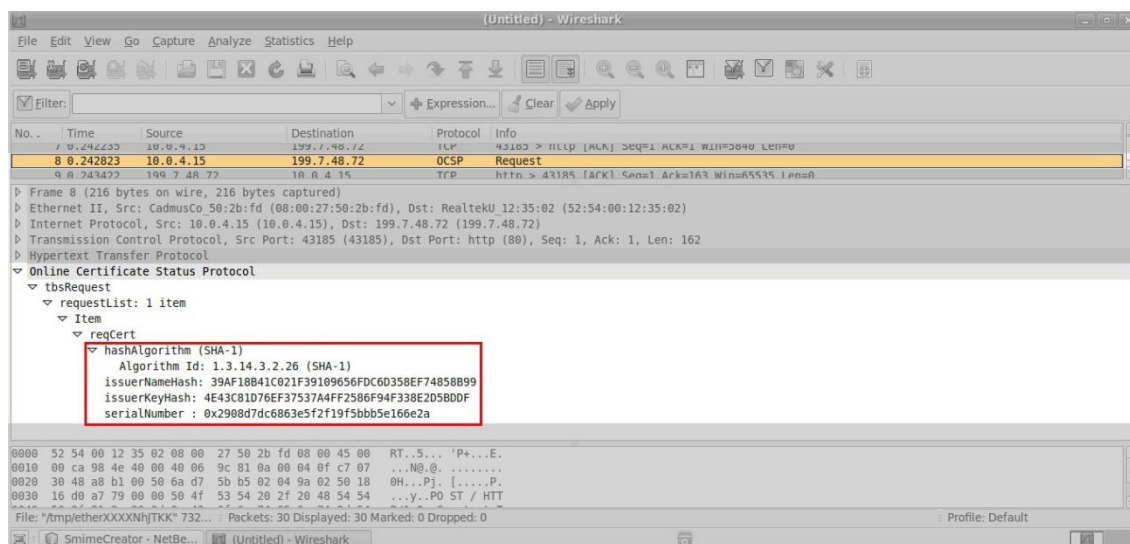
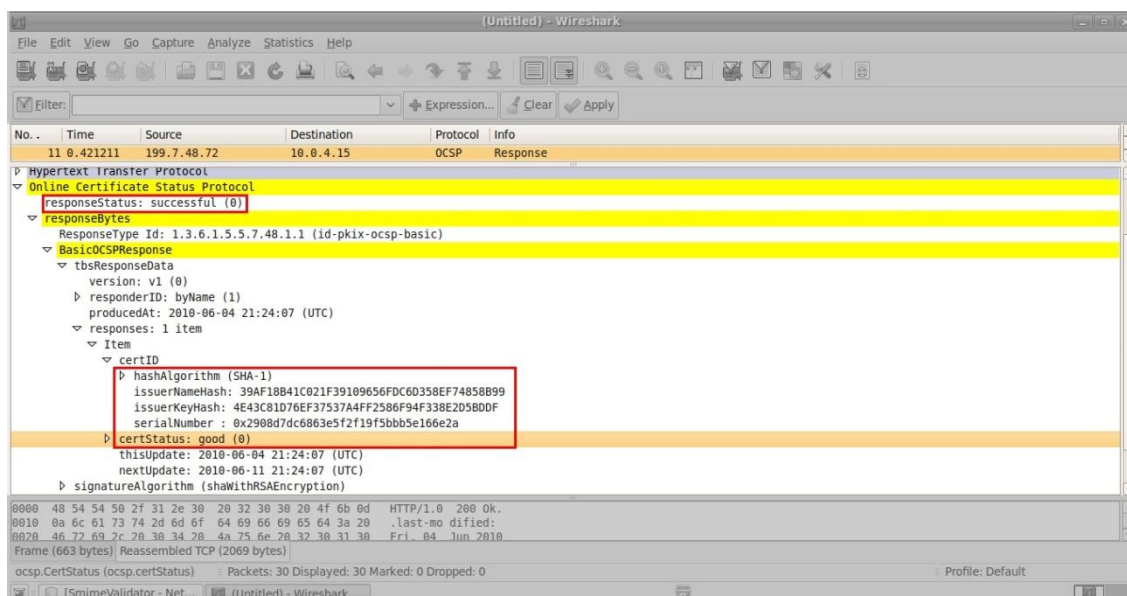


Figura 6.1 - Captura de pedido OCSP no *Wireshark*

Figura 6.2 - Captura da resposta OCSP no *WireShark*

No primeiro campo marcado a vermelho da Figura 6.2 pode identificar-se o estado da resposta, neste caso, retornou sucesso. No segundo campo está a resposta relativa à validade do certificado, contém a identificação do certificado (tal como enviado no pedido) e o estado do certificado, neste caso, o certificado está válido. Existem mais campos que não serão aqui analisados pois não apresentam relevância para a compreensão do trabalho desenvolvido nesta dissertação.

Tendo demonstrado o correcto funcionamento do OCSP até à comunicação HTTP, estão validados os campos de construção do pedido, envio deste e recepção da resposta. Na Figura 6.3 pode ver-se o resultado durante a execução do código. De acordo com o esperado na resposta analisada através do *WireShark*, o certificado retornou um estado válido.

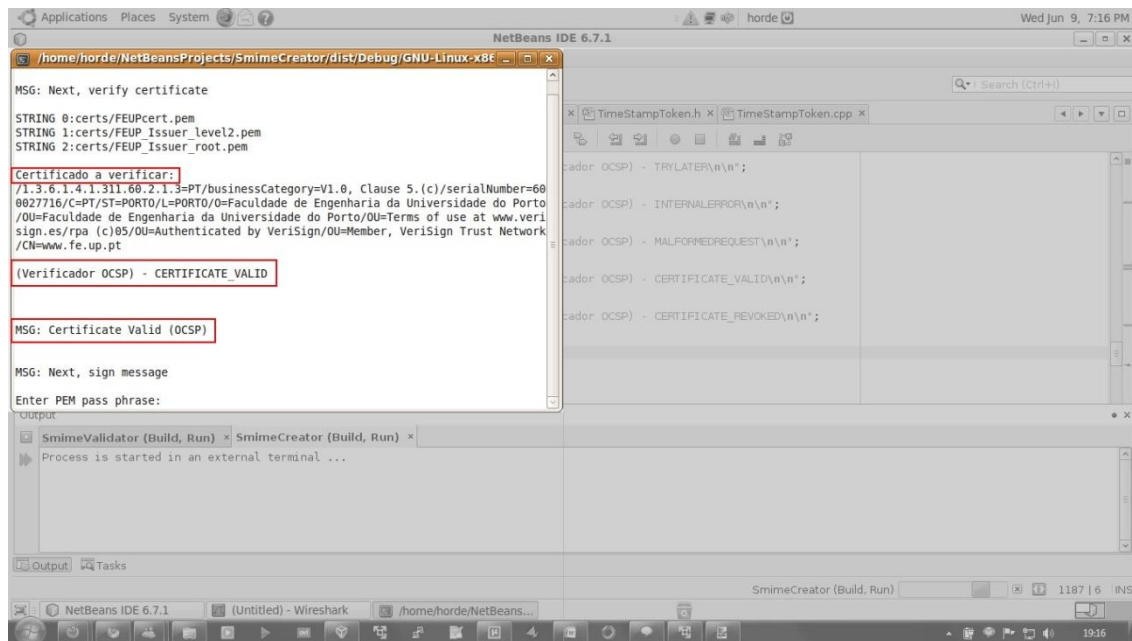


Figura 6.3 - Verificação de uma mensagem OCSP

6.1.1.2 CRL

O método adoptado para a demonstração do método por CRL é igual ao usado na demonstração do OCSP. Inicialmente analisa-se o tráfego de rede para ver o que é enviado e recebido, validando toda a construção do pedido e recepção da resposta. De seguida é feita a verificação de validade do certificado por comparação com a lista de certificados revogados. Na Figura 6.4 pode-se ver a captura relativa ao pedido, esta é simplesmente um pedido HTTP. A vermelho está marcada a lista pretendida. Na Figura 6.5 pode ver-se a captura relativa à resposta, o resultado que se vê é a agregação de todos os pacotes fragmentados recebidos. Assinalado a vermelho pode ver-se a lista de certificados revogados presentes na CRL.

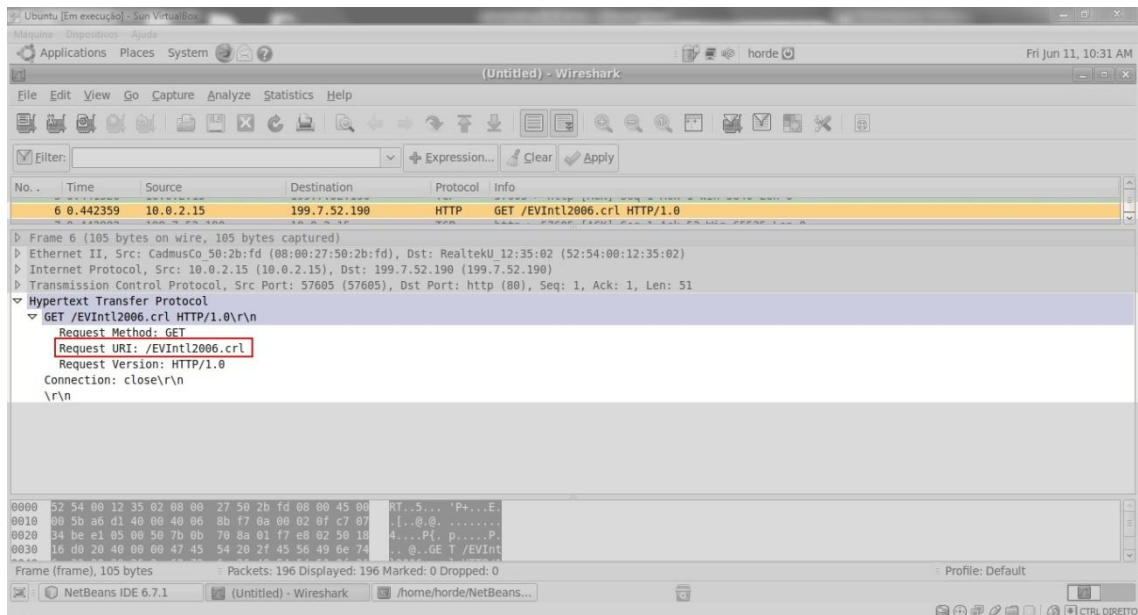


Figura 6.4 - Captura do pedido CRL no WireShark

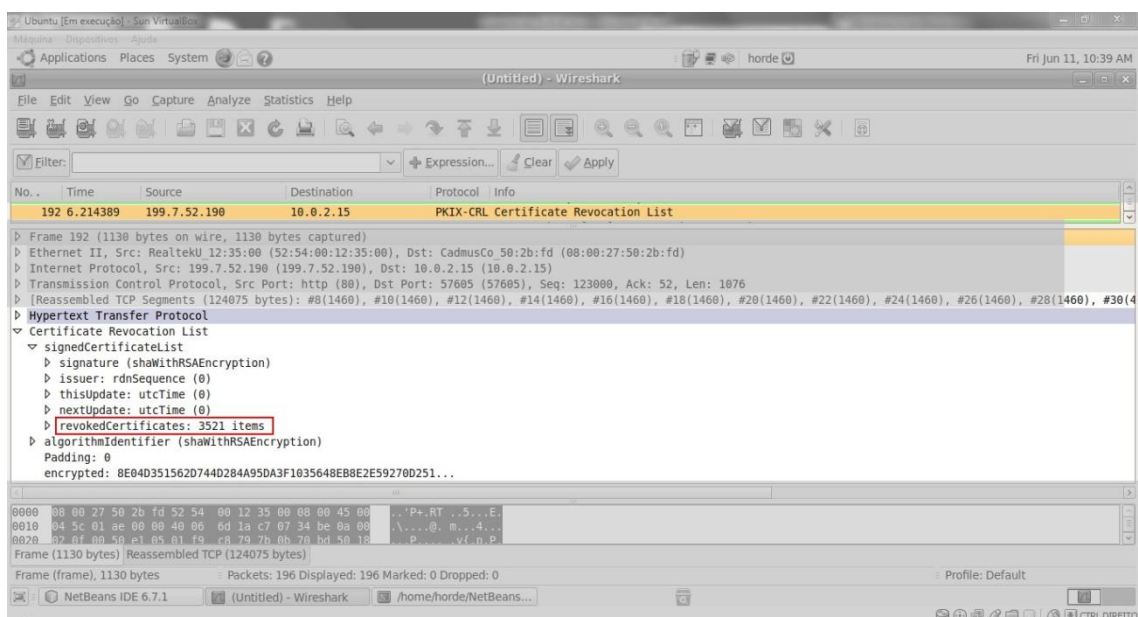


Figura 6.5 - Captura da resposta CRL no WireShark

Na Figura 6.6 mostra-se o resultado durante a execução do código, este procede à análise da validade do certificado usando a lista recebida. Forçou-se o uso do CRL desactivando o OCSP. O primeiro campo vermelho é respectivo à comunicação efectuada para a obtenção da lista. O segundo campo vermelho é a verificação do certificado usando a CRL recebida. O terceiro é o retorno do estado do certificado para o programa principal.

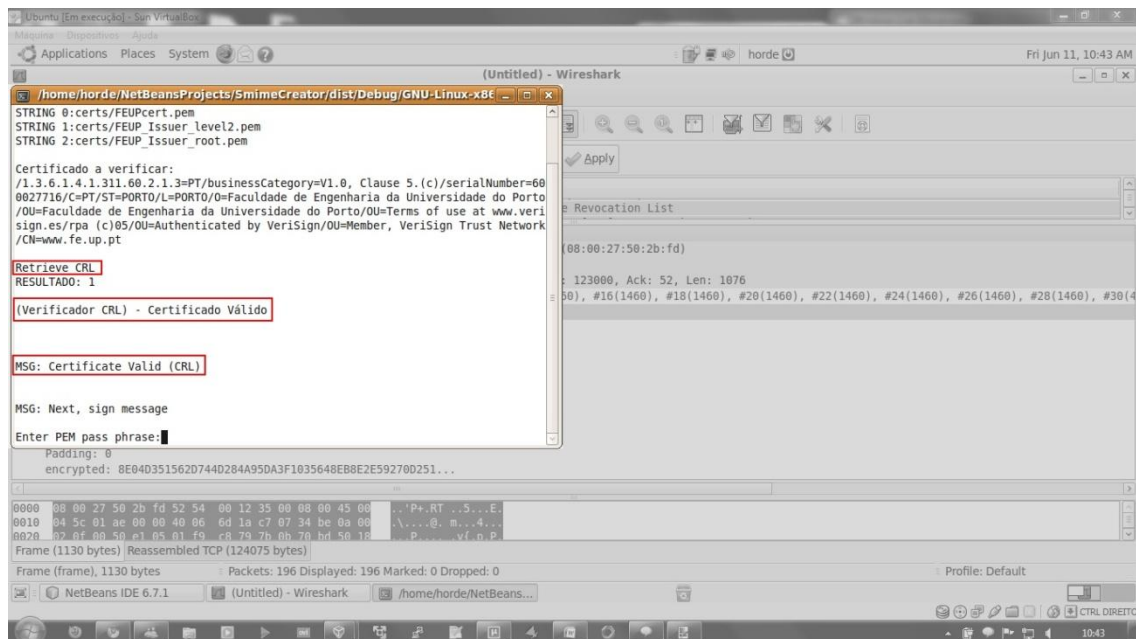


Figura 6.6 - Verificação de uma mensagem através de CRL

6.1.2 Assinatura

Para demonstrar a assinatura de mensagens, é apresentado o resultado da execução do programa, o ficheiro de origem e o ficheiro criado com o resultado da assinatura. Na Figura 6.7 está o resultado da execução do programa, marcado a vermelho pode ler-se que a assinatura foi criada com sucesso, onde está o ficheiro de origem e onde foi guardado o ficheiro com a mensagem assinada. Na Figura 6.8 vê-se do lado esquerdo a mensagem original e do lado direito a mensagem assinada.

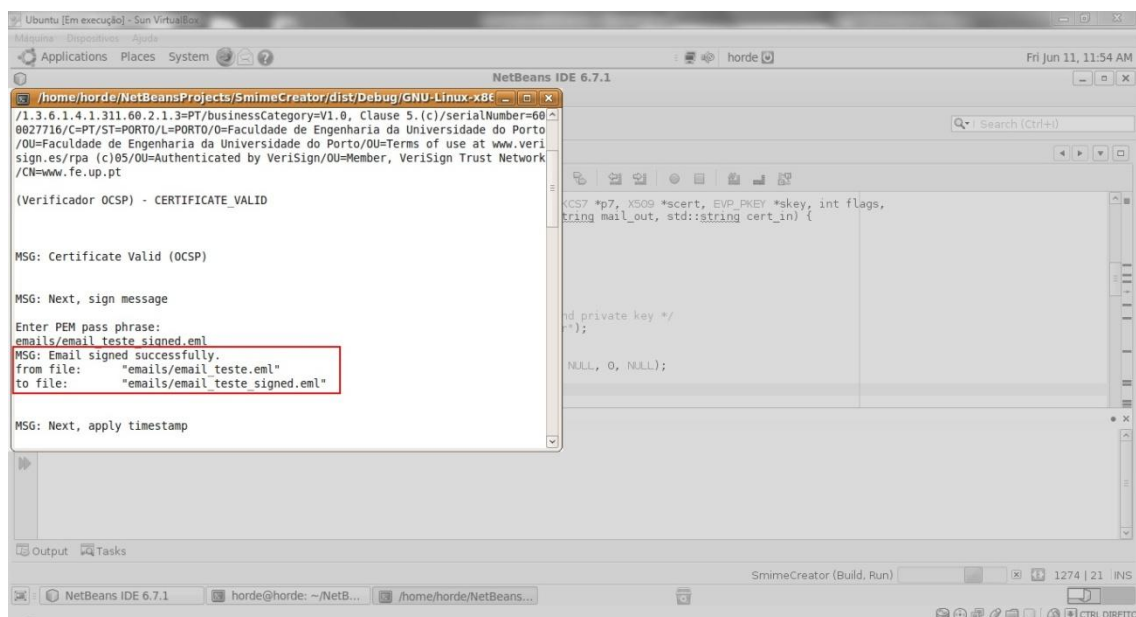


Figura 6.7 - Captura do resultado da assinatura de mensagens

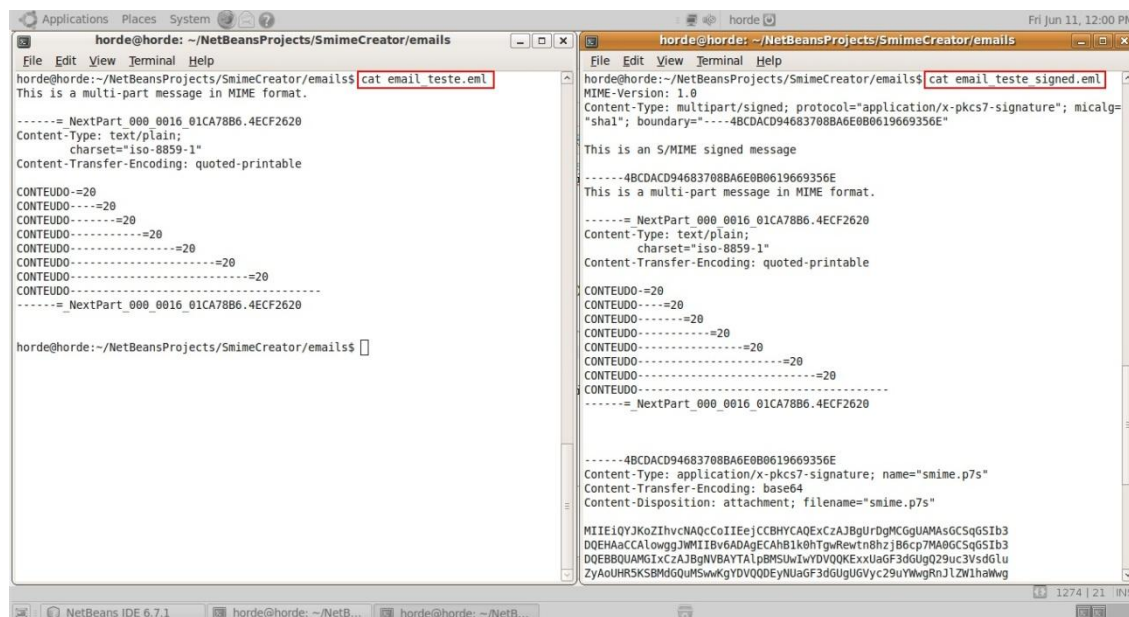


Figura 6.8 - Captura da mensagem original e da mensagem assinada

6.1.3 *Timestamping*

Para a demonstração do processo de *TimeStamping* no módulo de criação é adoptado um processo onde é demonstrada a comunicação com a TSA, mostrando assim o correcto funcionamento da criação do pedido e execução deste. A seguir é demonstrado o correcto processamento dos dados recebidos e analisada a resposta recebida. Após isto demonstra-se a aposição do TST à mensagem. Na Figura 6.9 está o pedido do timestamp à TSA e na Figura 6.10 está a resposta enviada pela TSA.

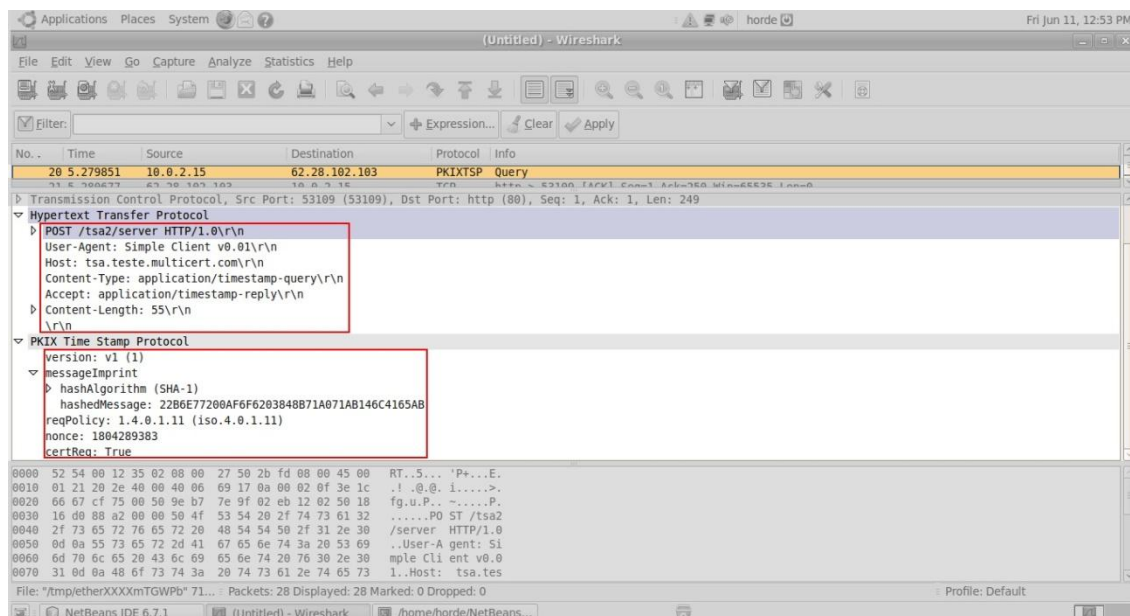


Figura 6.9 - Captura do pedido *timestamp* à TSA no WireShark

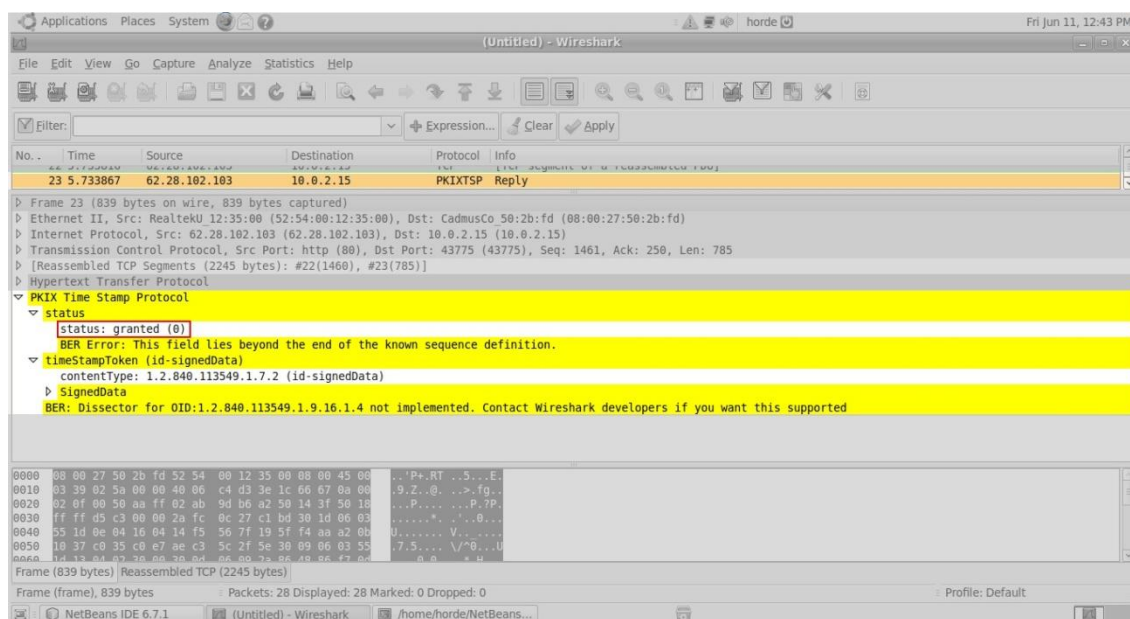


Figura 6.10 - Captura da resposta enviada pela TSA no WireShark

Dado que o *WireShark* não tem implementado o processamento dos campos relativos aos *timestamps*, parte dos campos não podem ser analisados neste, no entanto a resposta contém o estado 0 (*granted*). Para analisar o TST recebido na Figura 6.11, no primeiro campo a vermelho, pode ler-se o resultado do processamento da mensagem recebida e os campos desta. No segundo campo a vermelho, está o resultado da verificação do TSR. Neste caso o TSR é válido.

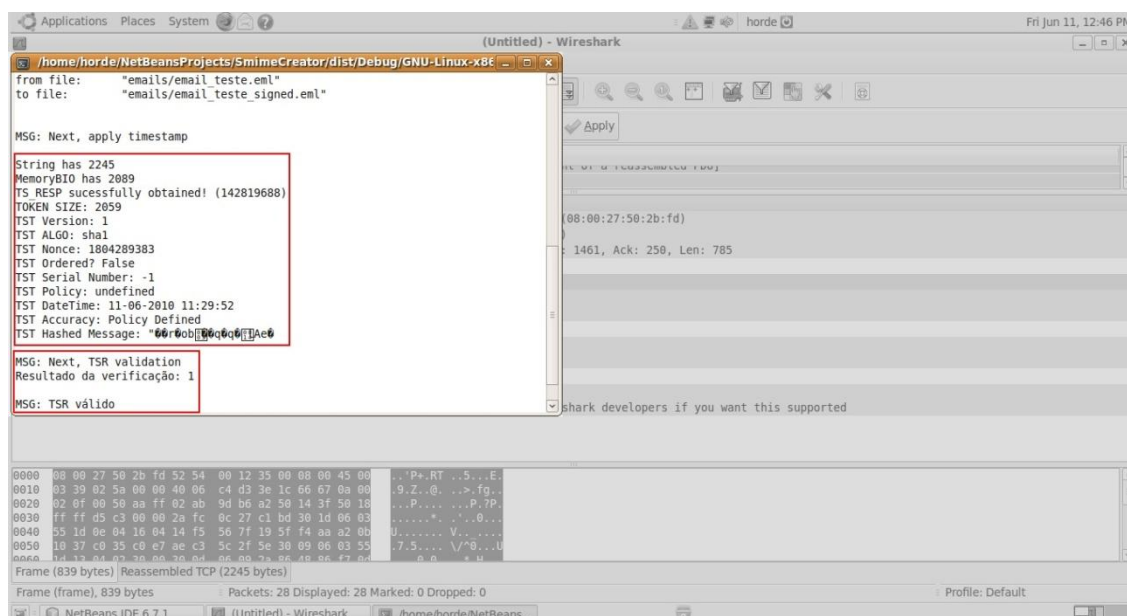


Figura 6.11 - Análise do TST e validação do TSR recebido

Sendo o TST válido e tendo em conta que a validação do TSR implica a validação do TST, procede-se ao anexo do TST à mensagem. Na Figura 6.12 pode ver-se o resultado da execução do código. Dado que o TST não poderia ser analisado pois está codificado com a assinatura, o tamanho em comparação com a mensagem assinada simples, mostra que o TST foi incluído na mensagem.

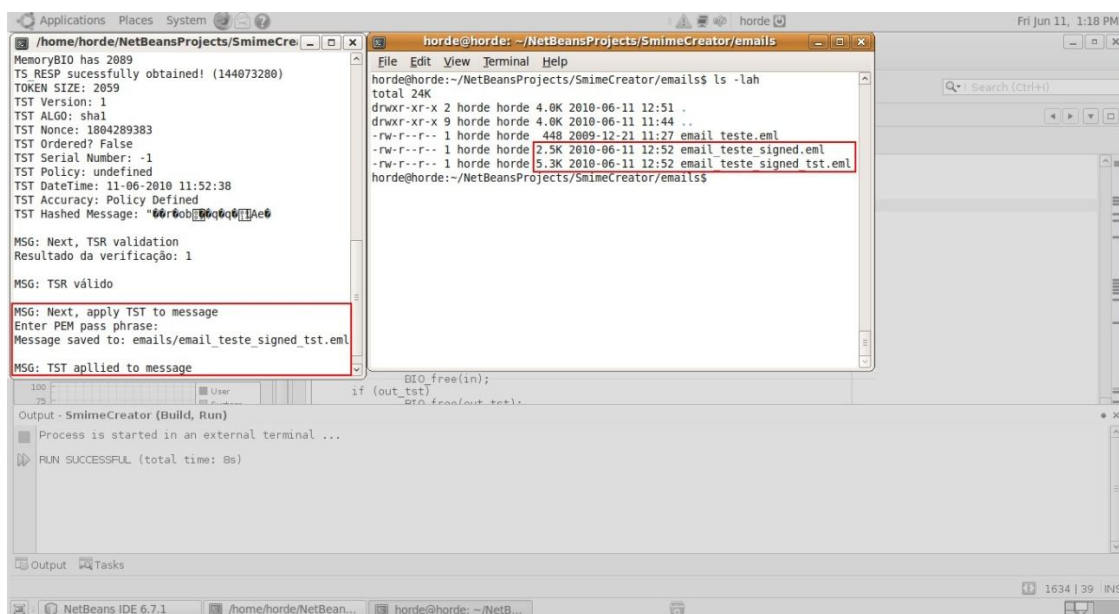


Figura 6.12 - Anexação do TST à mensagem

6.2 Módulo de Verificação

O módulo de verificação inclui a validação de certificados e análise de *timestamps*. As demonstrações serão efectuadas nesta ordem.

6.2.1 Verificação de Certificados

O código de validação de certificados no módulo de verificação é baseado no mesmo código que o do módulo de criação (6.1.1).

6.2.2 Timestamping

Para demonstrar o processamento de *timestamping* no módulo de verificação a demonstração está dividida em duas fases, inicialmente é demonstrada a pesquisa do TST na mensagem, e depois é demonstrado o processamento do TST e a devolução dos seus dados.

6.2.2.1 Pesquisa do TST na mensagem

Assume-se que, encontrando o TST, o módulo carregou correctamente a mensagem ficando assim também esse passo validado. A pesquisa do TST é efectuada recursivamente, na Figura 6.13 pode ver-se durante a execução do código, marcado a vermelho, a descoberta do TST na mensagem.

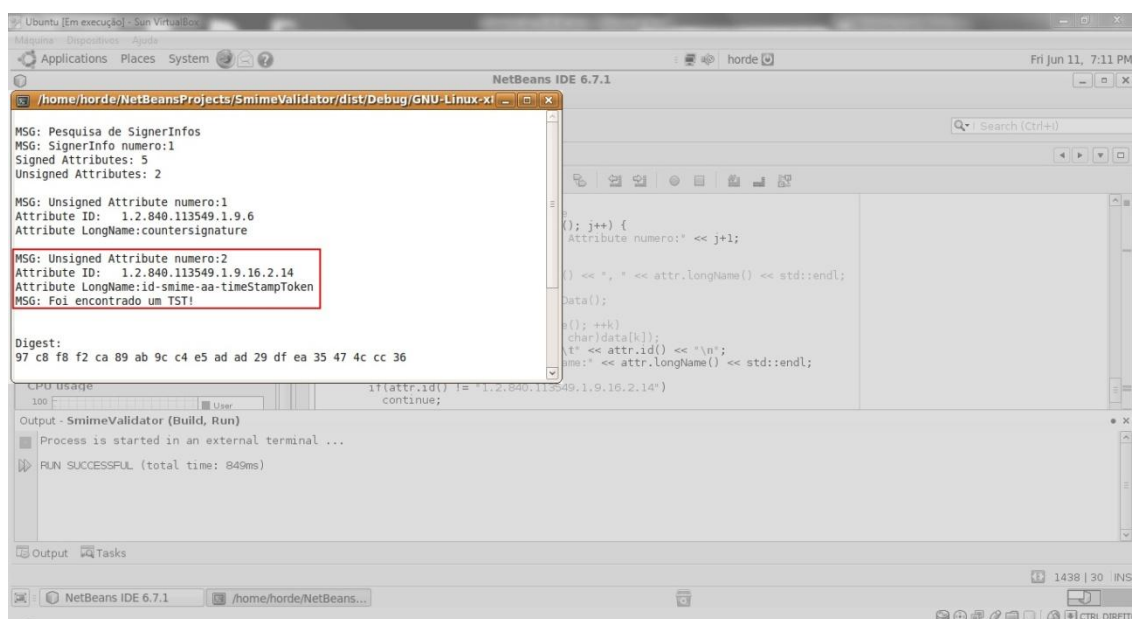


Figura 6.13 - Pesquisa do TST na mensagem

6.2.2.2 Análise do TST e retorno da data

Na Figura 6.14 encontra-se o resultado da análise do TST e comparação das *hashs*. O estado é retornado ao programa e é impressa a data em que foi criado o *timestamp*.

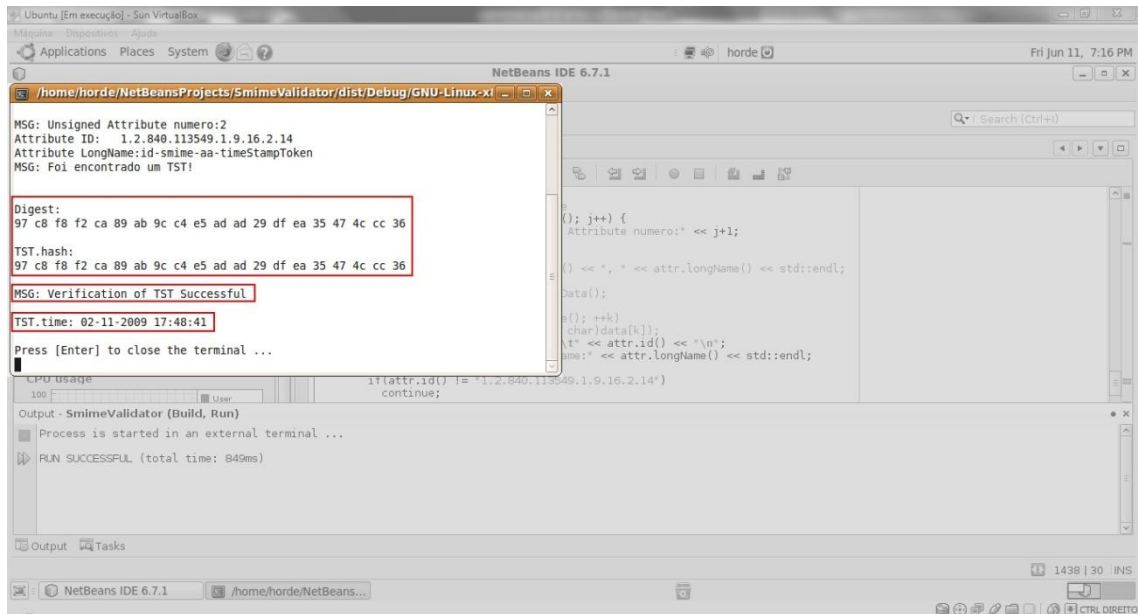


Figura 6.14 - Verificação do TST no módulo de verificação

6.3 Integração no *Horde* do módulo de verificação

A integração no *Horde* foi feita com a mudança da interface do IMP. Na Figura 6.15 está uma captura de imagem da interface do *Horde* após a modificação. Esta mudança à interface foi feita com a introdução de um botão em cima e em baixo para proceder à verificação do TST. Esta está assinalada a vermelho na figura.

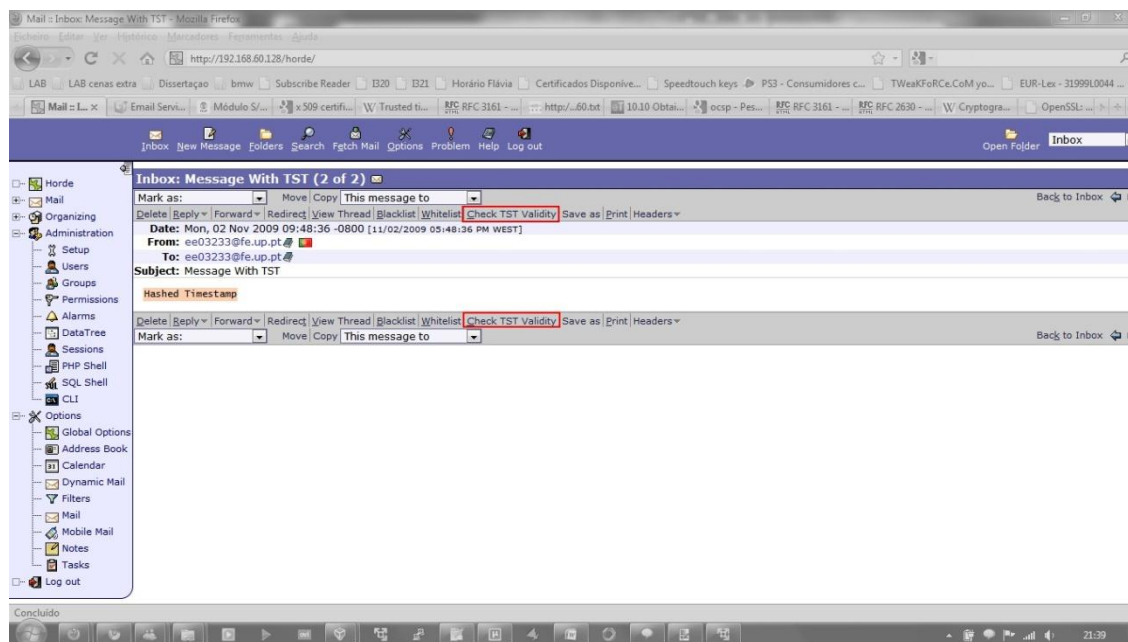


Figura 6.15 - Interface do *Horde* modificada

Ao pressionar qualquer um destes botões é invocado o módulo de verificação, cujos resultados são impressos numa janela “*pop-up*”. Um exemplo deste resultado pode ser visto na Figura 6.16. De forma a tornar o sistema mais amigável, a versão final deverá relacionar a data/hora directamente com a mensagem em vez do TST. Desta forma existirá uma melhor percepção do valor do sistema.

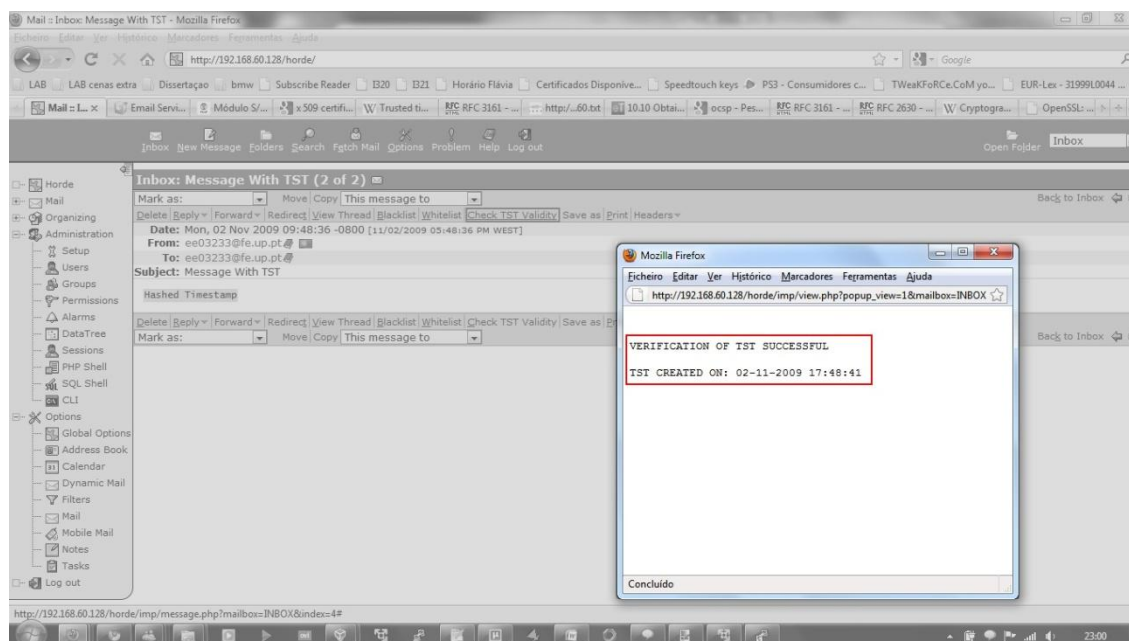


Figura 6.16 - Janela *pop-up* da verificação no *Horde*

6.4 Discussão

Os resultados procurados durante o desenvolvimento foram sempre resultados de demonstração de funcionamento. Procurou-se uma implementação de acordo com os RFCs e normas definidas de forma a tornar estes módulos o mais compatíveis possível. A dificuldade em avaliar resultados foi, em parte, a falta de implementações para comparação. Dado que não existia outra implementação no campo dos *timestamps*, os resultados apresentados pelo *OpenSSL* foram o objecto de uma primeira análise. Sempre que necessário foi investigada mais profundamente a implementação das funções do *OpenSSL* de forma a melhor se entender o que era processado durante a invocação destas.

Numa segunda iteração, sempre que havia dados disponíveis, estes eram analisados de forma manual. Esta análise foi executada com o intuito de confirmar que os campos presentes estavam preenchidos de acordo com o esperado. Uma das dificuldades nesta análise foi a grande diversidade de variáveis nas estruturas que continham os dados.

A integração no *Horde* acrescentou a este sistema uma funcionalidade não implementada até à data, acrescentando valor a um sistema tão usado como é o caso do *WebMail*.

Capítulo 7

Conclusões

Foi desenvolvida uma solução para *WebMail* que integra processamento de mensagens S/MIME acrescidas de *timestamps* criptográficos baseados no RFC 3161.

7.1 Contribuições e Resultados

Definiu-se e implementou-se uma solução de *WebMail* que integra S/MIME e *timestamping*. Esta solução fornece uma alternativa de email seguro mais amigável para o utilizador. Todo o processamento, certificados e chaves, estão do lado do servidor. Qualquer terminal pode aceder e enviar mensagens com garantias de integridade, autenticidade, não repúdio da origem e da data.

A solução desenvolvida segue a recomendação do RFC 3161 para aplicação de *timestamps* em S/MIME. A implementação consiste no desenvolvimento de dois módulos, um de criação e um de verificação. O módulo de criação é responsável por gerar mensagens S/MIME assinadas e com TSTs. O módulo de verificação processa as mensagens S/MIME, o TST e apresenta o resultado da validação na interface do *WebMail*. Foi escolhido o *Horde* como sistema de *WebMail* e o OpenSSL para as funções S/MIME e de *timestamping*.

7.2 Trabalho Futuro

No sentido de difundir a utilização de *timestamps*, deverão ser desenvolvidos *plug-ins* para clientes locais que permitam a verificação dos *timestamps* incluídos em mensagens

S/MIME. Deverá também ser desenvolvido um módulo para MTAs que permita a detecção de mensagens S/MIME e aposição de *timestamps*.

Referências

- [1] **Alma Whitten, J. D. Tygar.** Usability of Security: A Case Study. 1998.
- [2] *How to Time-Stamp a Digital Document.* **Stuart Haber, W. Scott Stornetta.** 1991.
- [3] OpenSSL: The Open Source toolkit for SSL/TLS . [Online] [Cited: 05 22, 2010.] <http://www.openssl.org/>.
- [4] **Stallings, William.** *Cryptography and Network Security - Principles and Practices - Fourth Edition.* s.l. : Pearson Education, 2006.
- [5] **Postel, Jonathan B.** RFC 821. *SIMPLE MAIL TRANSFER PROTOCOL.* 1982.
- [6] **N. Freed, N. Borenstein.** RFC 2045 e 2046. *Multipurpose Internet Mail Extensions.*
- [7] Mime Overview - Part I. [Online] Junho 6, 2010. <http://www.aspnetmime.com/help/welcome/overviewmimei.html>.
- [8] **B. Ramsdell, Editor.** RFC 2633. *S/MIME Version 3 Message Specification.* 1999.
- [9] **Moser, Heinrich.** S/MIME. 2002.
- [10] *Multi-Party Security System.* **L. J. Mary, S. P. Rajagopalan.** s.l. : Medwell Journals, 2007, Asian Journal of Information Technology.
- [11] **Comodo.** Code Signing FAQs. [Online] [Cited: Junho 18, 2010.] <http://www.instantssl.com/code-signing/code-signing-faq.html>.
- [12] MDDE. [Online] [Cited: Junho 18, 2010.] <https://sce.ctt.pt/mdde/index.html>.
- [13] **MultiCert, CTT e.** [Online] [Cited: Junho 18, 2010.] [https:// sce.ctt.pt /mdde /manual_validacao_mdde.pdf](https://sce.ctt.pt/mdde/manual_validacao_mdde.pdf).
- [14] *BouncyCastle.org.* [Online] [Cited: Maio 2010, 16.] <http://www.bouncycastle.org/>.

-
- [15] C. Adams, P. Cain, D. Pinkas, R. Zuccherato. RFC 3161. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. 2001.
- [16] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams. RFC 2560. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. 1999.
- [17] C. Adams, S. Farrell. RFC 2510. *Internet X.509 Public Key Infrastructure Certificate Management Protocols*. 1999.
- [18] Winkle, William Van. *SSL Certificate Validity*. [Artigo] s.l. : Processor, 2009.