TÉCNICO LISBOA

AISS Secure Mail

Serviço de Emails Seguros

Manual Técnico

57701: Gonçalo Carito

68210: Dário Nascimento

MERC

IST-TAGUSPARK

2012/2013

Conteúdo

Capítulo 1

Introdução

Este documento descreve a implementação de um Serviço de Emails Seguro. Esta implementação teve como premissa o pedido de um **cliente** para criar um plugin para o programa de emails *Mozilla Thunderbird*.
Este plugin teria que ter a possibilidade de garantir a **confidencialidade**, **não repudiação**, **autenticidade** e, como resultado de um requisito opcional por parte do Cliente, **garantia temporal** de uma mensagem trocada entre 2 utilizadores. Além disso, a garantia de *confidencialidade* teria de ser realizada através da Cifra/Decifra AES por uma caixa fornecida pelo Cliente e a *não repudiação* teria de ser assegurada através da assinatura da mensagem com o Cartão de Cidadão da República Portuguesa. O Cliente não exigiu quaisquer requísito extra em relação à *garantia temporal*, apenas que deve atribuir e validar um Timestamp Seguro à mensagem, sendo tal como as restantes operações: Cifra e Assinatura, de caracter opcional aquando da realização da operação.
Desta forma identificámos, no serviço a disponibilizar, 2 sujeitos principais:

- Sender - Tem como input uma pasta de dados que pode comprimir, cifrar, assinar e adicionar timestamp e gera um output para ser enviado ao destinatário (por exemplo por email)
- Receiver - Tem como input o email seguro do sender que irá decifrar, verificar assinatura, timestamp e descomprimir para receber o conteúdo original.

A solução apresentada cumpre os requisitos de segurança estabelecidos pelo Cliente: utilizando uma interface própria que gera um ficheiro que pode não só ser enviado

pelo cliente de email *Thunderbird* como por qualquer outro sistema de transferência de texto ou dados.

No **Capítulo 2** é apresentada a Arquitectura do Sistema constituido pelo módulo cliente (Sender ou Receiver) onde são implementadas as funcionalidades de cifra, assinatura e compressão, e pelo servidor de timestamp seguro.
No **Capítulo 3** é feita uma Avaliação à performance e desempenho do sistema, tendo em conta as várias operações e a sua contextualização em cenário real.
No **Capítulo 4** são tiradas as conclusões sobre o trabalho efectuado e respectivas reflexões.

Capítulo 2

Arquitectura

Neste capítulo é apresentada a arquitectura do sistema, a interligação entre os vários módulos e uma explicação detalhada do funcionamento dos vários procedimentos como a cifra e assinatura.

2.1 Visão Geral

A solução desenvolvida permite a troca de mensagens de modo seguro e com garantias temporais. Esta solução é constituída por 3 módulos: Sender, Receiver e um servidor de Timestamp (**Figura 2.1**).

O Sender pretende enviar um conjunto de dados para o Receiver mas com garantias de **confidencialidade**; **autenticidade** e **não repudiação**; instante temporal. Estas garantias são dadas por mecanismos externos: CipherBox (caixa de cifra ethernet), Cartão Cidadão Português e Servidor Timestamp, respectivamente. Através do mecanismo de hash garante-se, também, **integridade**.

Num cenário de utilização, o Sender selecciona a pasta que pretende enviar e escolhe cada uma das seguintes acções:

- Sign - assinar o(s) ficheiro(s) com o seu Cartão de Cidadão da República Portuguesa
- Timestamp - adicionar um timestamp seguro assinado por uma Autoridade Cer-

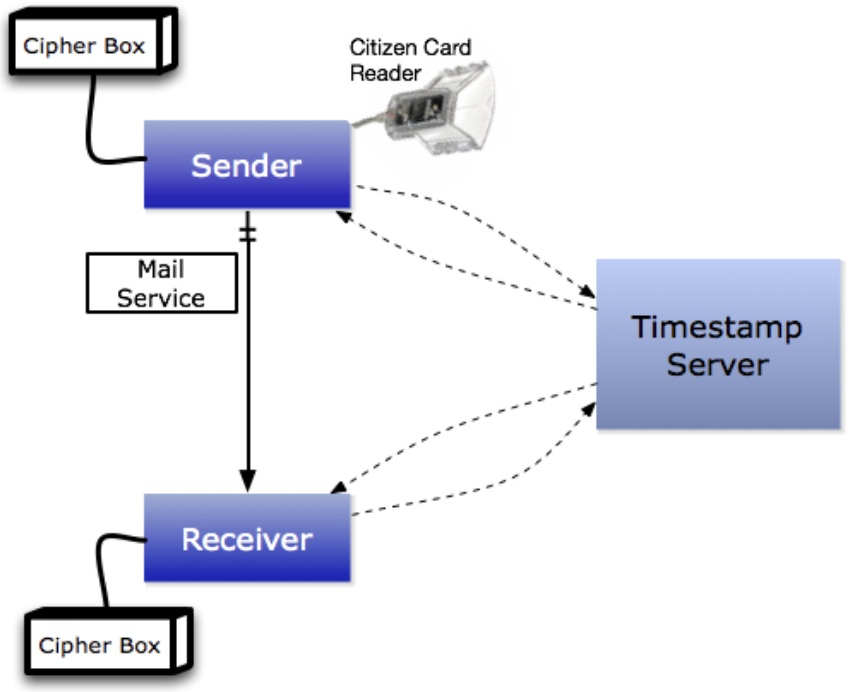

Figura 2.1: Arquitectura do Sistema

tificada

- Cipher - cifrar o(s) ficheiro(s)

Supondo que todas as opções foram seleccionadas é realizada a cadeia de acções descrita na **Figura 2.2**.

Os ficheiros seleccionados são comprimidos num único. De seguida, este ficheiro é assinado recorrendo ao Cartão de Cidadão e o seu *hash* é enviado ao Servidor de Timestamp Seguro. O servidor cria um par - associando um *instante temporal* T ao *hash*. Este par é assinado pelo servidor de modo a garantir que o servidor recebeu o *hash* naquele instante. Este certificado de Timestamp é devolvido ao Sender.
A assinatura de cartão de cidadão, o certificado de Timestamp e os dados são organizados numa estrutura e esta estrutura é cifrada. Os dados cifrados são colocados dentro de uma nova estrutura que descreve as operações realizadas e permite inverter o processo. A estrutura é então codificada e escrita em disco em formato de ficheiro para que o Sender possa enviar.

Quando o Receiver recebe este ficheiro protegido efectua as operações pela ordem

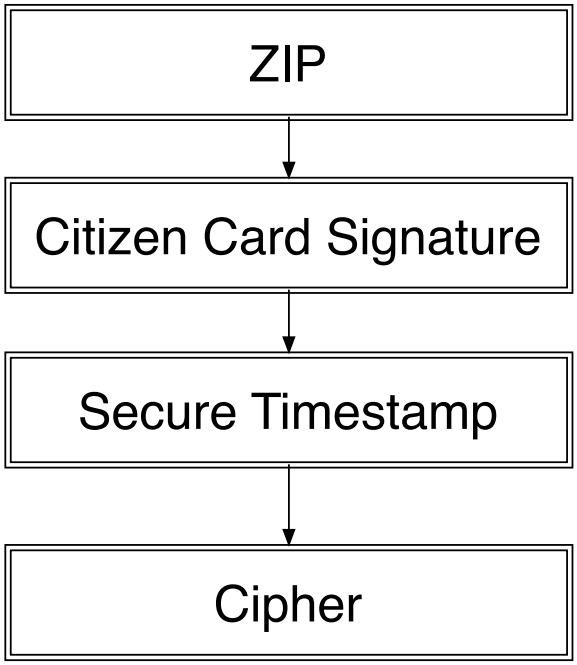

Figura 2.2: Ordem de execução

inversa, começando pela decifra, verificação da validade do Timestamp através do servidor, verificação da Assinatura e, termina, descomprimindo os dados.

2.2 Zipping/Unzipping

Sempre que o Sender envia uma mensagem, seleciona a pasta que pretende enviar. Esta pasta é comprimida utilizando a biblioteca *Zip* nativa do Java 6. O output desta operação é um ficheiro único que contém toda a pasta comprimida e por isso reduz a dimensão do anexo a cifrar, assinar e enviar: $ZippedFile = Zip(file_1 + ... + file_n)$
É então criada uma estrutura que será preenchida nos próximos passos com os metadados.

O Receiver ao receber a mensagem - e depois de efectuar os restantes passos necessários - efectua o Unzip retornando da pasta comprimida aos ficheiros originais.

2.3 Sign/Verify

A segunda operação a ser executada pelo Sender é a operação de Assinatura; no caso do Receiver esta é a penultima operação a ser executada (antes do unzip).

Para garantir a **não repudiação** e **autenticidade** do ficheiro comprimido é utilizado o Cartão do Cidadão através da biblioteca PKCS11. O ficheiro é *hashed* com SHA-1 e assinado com a chave RSA privada contida no Cartão de Cidadão: $Assinatura = \{ZippedFile, H(ZippedFile\}K_s$ com $H(x)$ a hash de x e K_s a chave privada.
A assinatura e o Certificado de chave pública do cartão de cidadão são colocados na estrutura de metadados.

Do lado do Receiver, a assinatura é verificada em 2 passos. Primeiro verifica se o certificado enviado é válido, verificando a assinatura do certificado com os *root certificate* do Estado Português que são incluídos na solução. De seguida, verifica se a mensagem foi assinada pelo detentor do certificado. Caso isto se verifique, é apresentado o nome do detentor do certificado.
Este mecanismo possui uma optimização por assinar apenas o contéudo comprimido. Não obstante, **garantimos apenas a não repudiação do contéudo comprimido e não do contéudo original.**

2.4 Timestamp Seguro

Para utilizar um mecanismo de Timestamp seguro, o Sender envia ao Servidor de Timestamp um pedido com o hash do(s) ficheiro(s) previamente comprimido(s).
O servicor cria um Timestamp T com a data actual e anexa-o ao *hash* recebido. Este par é assinado com a chave privada do servidor, criando uma mensagem com o formato: $\{(H(m), T), T\}K_s$ em que $H(m)$ é a hash da mensagem, T o timestamp e K_s a chave privada do Servidor de Timestamp.
Este serviço apenas garante que **este hash foi assinado pelo servidor na data definida.**

O Receiver verifica a validade do timestamp, ou seja, o momento em que o servidor de confiança associou o hash da mensagem. Para tal, é utilizado o certificado do Servidor de Timestamp seguro que está incluido no pacote da solução.
Não é dada qualquer garantia da data de envio de email porque o utilizador pode enviar o mail *a posteriori*.

2.5 Cipher/Decipher

A última operação a ser executada pelo Sender e a primeira a ser executada pelo Receiver é a operação de cifra (e decifra, respectivamente). Para efectuar a cifra, o Sender utiliza a Caixa de Cifra via ethernet fornecida pelo Cliente. O software base para conecção com a caixa foi disponibilizado em linguagem C. As invocações Java foram convertidas para invocações C através do JNI. A mensagem é cifrada por blocos, o que possibilita a cifra de ficheiros de grandes dimensões. O Receiver utiliza também uma Caixa de Cifra ethernet para decifrar a mensagem.
Esta caixa realiza cifra e decifra AES. A chave utilizada está contida na caixa por isso esta caixa constitui um mecanismo de segredo partilhado, ou seja, apenas quem tem a caixa pode decifrar o contéudo.

2.6 Base64

Antes de salvar (ou enviar para o email) o ficheiro gerado é codificado em base64. Este sistema codifica cada byte em 6 bits. Estes 6 bits são caractéres conhecidos pela norma ASCII e, por isso, o ficheiro pode ser enviado em formato de *plain-text* sem que alguns sistemas de codificação e *sanitize* removam caracteres desconhecidos e deste modo adulterem o contéudo da mensagem.

Capítulo 3

Avaliação

De modo a confirmar a segurança do nosso sistema, adulteramos as mensagens e assinaturas realizadas. Confirmarmos que o sistema detecta corretamente a adulteração de mensagens.
Foram realizados vários testes de desempenho do sistema utilizando um computador de utilizador comum MacBook Pro com processador 2.9 GHz Intel Core i7, memória 8 GB 1600 MHz DDR3 e sistema operativo MacOsX 10.8.3 e disco SSD (evidênciado na latência de escrita em disco). O servidor de timestamp foi executado localmente pelo que não são considerados atrasos na rede.
Foram realizadas 5 amostras por cada caso e determinados os níveis de confiança a 95% para os tempos obtidos. O directório enviado foi constituído por 286 ficheiros PDF e Word num total de 100MBytes.

- **Compressão:** A compressão do ficheiro converte $100MB$ em $95.5MB$. Esta taxa depende do formato e contéudo dos ficheiros de origem. Demorou em média $4484ms$ com desvio padrão de $30ms$.
- **Hashing do ficheiro:** A realização do hash do ficheiro comprimido para utilizar nas assinaturas demora em média $700ms$ com desvio padrão de $10ms$.
- **Assinatura com cartão de cidadão:** A assinatura com cartão de cidadão demorou $936ms$ com desvio padrão de $24ms$.
- **Timestamp:** O serviço de timestamp demora $70ms$ a decorrer com desvio padrão de $5ms$.

- **Codificação e escrita em disco:** A conversão para base 64 e escrita em disco demorou $1335ms$ com desvio padrão de $35ms$. A codificação de base 64 implica um aumento da dimensão do ficheiro ao factor de $8/6$ pelo que o ficheiro final tem $130.7MB$.

- **Descompressão e escrita em disco**A descompressão demorou $1380ms$ com desvio padrão $32ms$.

- **Verificação timestamp:** A verificação do timestamp demorou $32ms$ com desvio padrão de $6ms$.

- **Verificação Assinatura:** A verificação da Assinatura demorou $6ms$ com desvio padrão de $3ms$.

- **Cifra e Decifra:** A especificação técnica da caixa fornecida pelo cliente indica $60segundos$ para cada $100MB$, pelo que assumimos este valor na ausência da possibilidade de realização de testes experimentais.

No total, o processo de envio demora (excluindo a cifra) **6 segundos** enquanto que o processo no receptor demora **3 segundos**. Estes valores não são significativos para a experiência do utilizador quando comparados com o tempo de inserção do PIN do cartão de cidadão e da cifra pela caixa fornecida pelo cliente.
A excelente performance do nosso sistema deve-se em parte ao facto de os dados só serem lidos e escritos uma única vez no disco. O reduzido numero de acessos a disco reduz o atraso do processo. Por outro lado implica a utilização de mais memória RAM do utilizador. Este processo teve um pico de utilização máxima de memória de 300MB. Consideramos que este custo não é significativo dadas as especificações atuais dos computadores para utilizador final. Caso o utilizador tenha menos memória, a solução é aplicável mas para ficheiros de menores dimensões.
A análise de performance conclui portanto que a nossa solução é eficiente.

Capítulo 4

Conclusão

O objectivo principapal neste trabalho prendeu-se na garantia das características de segurança exigidas pelo Cliente, através dos seus requísitos funcionais, nomeadamente:

- Confidencialidade
- Autenticidade e Não-Repudiação
- Garantia Temporal

Ao efectuar a **cifra** de todo o conteúdo enviado pelo Sender, inserido previamente num objecto com metadados indicativos, garantimos a absoluta confidencialidade do conteúdo. A cifra ser realizada através do mecanismo de caixa, cria *per si* a garantia de partilha de um segredo exclusivo, a chave simétrica, indicando que apenas quem tem a caixa cifra/decifra a mensagem. A nossa solução integra as invocações Java integradas no restante código com as em Linguagem C presentes na interface da caixa, permitindo assim uma boa utilização da mesma.
Em fase de desenvolvimento do projecto, começamos por cifrar a totalidade da mensagem, o que apresentava problemas de memória - em particular na *heap* do Java, limitando o tamanho da mensagem a ser cifrada. No entanto, na nossa solução final, efectuamos cifra bloco a bloco que anula os problemas de memória permitindo a cifra de mensagens com, por exemplo, anexos de elevada dimensão.

A utilização do Cartão de Cidadão como mecanismo de garantia de **autenticidade** e **não-repudiação** revelou bastantes aspectos positivos. Não obstante o não-controlável

facto de o Cartão de Cidadão ser roubado juntamente com o PIN, este apresenta-se com um mecanismo viável no que toca a mecanismos de assinatura. O certificado que utilizámos - Certificado de Autenticação - valida os requisitos necessários não tendo o problema dos contornos legais que existem com o Certificado de Assinatura Digital, reconhecido pelo Estado como um meio legal para assinar documentos, comportando-se - em termos de implementação - de forma similar.
Apesar de a biblioteca do Cartão de Cidadão, *ptlib*, não ser muito estável, não nos trouxe problemas pois serve apenas na ligação entre Cartão e Certificados com o computador, sendo que para toda a gestão, assinatura e validação do processo, a biblioteca utilizada foi a *PKCS11* que é bastante estável. A não disponibilização da chave de cifra privada fora do Cartão de Cidadão invalida a sua manipulação mas não a sua utilização. Não obstante o facto de na nossa proposta de solução não existir necessidade de manipulação da mesma - é sempre um facto a reter. Na nossa solução, o certificado de chave pública do Cartão de Cidadão é enviado encapsulado no objecto final (se nos abstraírmos do facto de a mensagem poder ir cifrada ou não) que é enviado para o Receiver, juntamente com os metadados.

A **garantia temporal** não foi um requisito obrigatório por parte do Cliente, nem a sua forma de implementação. A utilização de um servidor de timestamp revelou ser uma boa escolha pois permitiu testes de exclusividade à operação do mesmo (sem consideração de atrasos na rede). O servidor de timestamp é, também, uma entidade de confiança entre ambos os módulos: Sender e Receiver. Daí a opção de ser enviado o hash da mensagem para o servidor, de forma a que este garanta a data da altura da assinatura. Não é possivel, na nossa solução, garantir data de envio do próprio email. No entanto, dada a transversalidade do nosso serviço - ou seja, a possibilidade de envio de utilização do conteúdo protegido por outros serviços - a garantia da data sobre o próprio conteúdo aparenta ser uma mais-valia importante.

Para além das características de segurança assinaladas existem, também, dois pontos a referir:

- Integridade
- Base 64

Apesar da **integridade** não ser, mesmo que indirectamente, um requisito do Cliente, o nosso sistema permite a garantia da mesma pois é utilizado o mecanismo de *hash* de relevante importância em operações de segurança. O mesmo pode ser aplicado à utilização da **Base 64** que permite que não haja restrições nos serviços a utilizarem a nossa soluçaõ, caso o anexo com o ficheiro protegido seja, por exemplo, enviado como *plaintext* como é o caso de alguns serviços de email que assim procedem.

No ínicio do trabalho, identificámos alguns desafios como:

- Possíveis problemas de memória
- Necessidade de assegurar eficiência nas operações
- Cartão de Cidadão com uma biblioteca instável

Tal como descrito ao longo desta conclusão, estes desafios forma ultrapassados aquando da necessidade de cumprir os requisitos principais de segurança.

Para além dos principais objectivos, também foram identificados outros como:

- Ter uma interface user-friendly
- Ser um sistema compatível com aplicações que acedem a ficheiros
- Manter uma boa performance

Para obtermos uma interface clean e de fácil utilização pelo cliente, integrada de forma natural com a *core* do trabalho, foi optada a utilização da biblioteca gráfica *swing* do Java. Como tal, foi concebida uma interface que permite uma boa ligação ao sistema de ficheiros, tornando-a facilmente interligável com outros programas de envio de ficheiros. De forma a cumprir e fornecer um serviço do mais possivel do agrado do Cliente, a aplicação apresenta por *default* a anexação do ficheiro protegido a uma mensagem no *Mozilla Thunderbird*. Trabalho futuro para permitir maior flexibilidade de escolha, passaria pela oferta de uma opção de interacção estrita com outros serviços como o *Gmail*, *Dropbox*, *Google Drive* entre outros.

Last but not least, a boa performance do programa foi atingida, sendo a nossa solução eficiente, tal como pode ser verificado com mais detalhe no Capítulo 3.