

Faculdade de Engenharia da Universidade do Porto



Implementação e validação de um modelo de certificação e distribuição de chaves para autenticação de sessões TLS baseado em DNSSEC

Valter Manuel Martins Mónica

Tese submetida no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major de Telecomunicações

Orientador: Prof. Dr. José Ruela

Co-orientador: Eng. Jaime Dias

Julho de 2008

Resumo

As comunicações através da *Internet* são por natureza inseguras. Há uma crescente necessidade de comunicações seguras, para as quais é necessária a autenticação das entidades, e confidencialidade e integridade dos dados. Muitas das soluções existentes, de entre as quais o SSL/TLS, dependem do PKIX, o qual recorre a certificados X.509 emitidos, tipicamente, por Autoridades de Certificação comerciais. A emissão dos certificados tem custos económicos e administrativos. Devido aos custos, alguns servidores *Web* usam certificados *self-signed*, os quais não permitem uma autenticação transparente. As CA comerciais não têm autoridade sobre as entidades que certificam, desempenham o papel de notário e definem as próprias políticas dos serviços que fornecem muitas vezes limitando as suas responsabilidades.

O DNSSec permite garantir a autenticidade e integridade dos registos DNS. Apesar de oferecer protecção contra muitas ameaças, até à data, isoladamente, não tem justificado o investimento e sua manutenção, o que tem impedido a sua implementação em larga escala. Esta tese apresenta uma solução com base num novo modelo de certificação, o ScalSec, que é baseada no DNSSec. É definido um novo registo para o DNSSec que permite adicionar um resumo de chave por nó. Cada nó assume-se como CA local e emite certificados para cada serviço. Ao DNSSec é adicionado apenas o resumo que representa o nó. O cliente recorre aos certificados X.509 adaptados e ao DNSSec, juntamente com os novos registos por nó, para autenticar o servidor. Suporta-se sinónimos DNS de forma transparente, sem modificações nos certificados.

Foi melhorado um cliente TLS, o do projecto *Mozilla*, comum ao *Firefox* e ao *Thunderbird*, clientes *Web* e de email, respectivamente. Foram também criados scripts que permitem a gestão automática dos certificados num nó, bem como a actualização dos respectivos registos no DNS.

Abstract

The communications over the Internet are by nature insecure. There is a growing need for secure communications, for which it is necessary to authenticate the entities, and confidentiality and integrity of data. Many of the existing solutions, including the SSL/TLS, depend on PKIX, which uses X.509 certificates issued, typically, by commercial Certification Authorities. The issuance of certificates has economic and administrative costs. Due to cost reasons, some Web servers use self-signed certificates, which do not allow a transparent authentication. The commercial CAs do not have authority over the certified entities, playing the role of notary and define their own policies for services they provide often limiting their responsibilities.

The DNSSec ensures the authenticity and integrity of DNS records. Despite it offers protection against many threats, so far, alone, has not justified the investment and maintenance, which has prevented its implementation on a large scale. This thesis presents a solution based on a new model of certification, the ScalSec, which is based on DNSSec. A new resource record for the DNSSec is defined which allows a hash of a key per node to be added. Each node takes itself as local CA and issues certificates for each service. In DNSSec is added only the hash that represents the node. The client uses the X.509 adapted certificates and the DNSSec, together with new resource records per node, to authenticate the server. It supports DNS CNAME resource records in a transparent manner, without modifications to the certificates.

A TLS client was improved, from the Mozilla project, common to Firefox and Thunderbird, Web and e-mail clients, respectively. We have also created scripts that allow the automatic management of the certificates in a node and the update of the relevant records in DNS.

Conteúdo

1	Introdução	1
1.1	Enquadramento	1
1.2	Modelo de certificação ScalSec	2
1.3	Arquitectura da implementação	2
1.3.1	DNSSec	3
1.3.2	Servidores	4
1.3.3	Cliente	4
1.4	Objectivos e contribuições relevantes	5
1.5	Estrutura da tese	6
2	Estado da arte	7
2.1	SSL/TLS	7
2.2	PKIX	8
2.3	PGP e SPKI/SDSI	11
2.4	DNSSec	11
2.4.1	Chaves de aplicação	12
2.4.2	Certificados	13
3	DNSSec	15
3.1	Adaptação do DNSSec	15
3.2	Configuração do DNSSec	17
3.3	Actualização dinâmica do DNSSec	19
3.4	Conclusões	20
4	Servidores	21
4.1	A CA que representa o nó	21
4.2	Serviços	23
4.3	Optimização e automatização	24
4.3.1	Variáveis a considerar	25
4.3.2	Variáveis de estado	25
4.3.3	Actualização dinâmica no DNS	26
4.3.4	Gestão e agendamento temporal	26
4.4	Conclusões	28
5	Cliente	29
5.1	Cliente TLS do Projecto Mozilla	29
5.2	Verificação dos certificados X.509	30
5.3	Validação do mapeamento do nome/endereço no nó servidor	33

5.3.1	LWRES	34
5.4	Interface com o utilizador	34
5.5	Compilação e instalação dos Clientes	35
5.6	Instalação do LWRES	37
5.7	Conclusão	39
6	Análise de resultados	41
6.1	Desempenho	41
6.2	Tamanho dos certificados transferidos	45
6.3	Conclusões	49
7	Conclusões	51
7.1	Revisão do trabalho desenvolvido	51
7.2	Contribuições relevantes	52
7.3	Trabalho futuro	52
A	DNSSec	53
A.1	Ficheiros das zonas DNS configuradas	53
A.1.1	db.porto.pt	53
A.1.2	db.10.16.172.in-addr.arpa	53
A.1.3	db.pt	54
A.1.4	db.16.172.in-addr.arpa	54
A.2	Exemplo da zona porto.pt assinada	55
B	Servidores	59
B.1	Ficheiros de configuração para OpenSSL	59
B.1.1	Exemplo de ficheiro de configuração para a CA do nó	59
B.1.2	Exemplo de ficheiro de configuração para serviço	60
B.2	Exemplos de certificados obtidos	60
B.2.1	Exemplo de um certificado obtido para a CA do nó	60
B.2.2	Exemplo de um certificado obtido para um serviço	61
C	Cliente	63
C.1	Configuração do LWRES	63
C.2	Código alterado	63
C.2.1	Alterações no ficheiro sslauth.c	63
C.2.2	Alterações no ficheiro certvfy.c	65
	Referências	74

Lista de Figuras

1.1	Modelo de certificação ScalSec.	3
1.2	Arquitectura geral da implementação do novo modelo.	4
3.1	Estrutura da cadeia de confiança das zonas configuradas para a implementação do DNSSec.	18
4.1	Estrutura da CA do nó e respectivos serviços.	22
5.1	Relação das bibliotecas para a implementação do cliente TLS.	30
5.2	Exemplo de erro de validação com ScalSec no Firefox.	35
5.3	Exemplo de sucesso na validação com ScalSec no Firefox.	36
5.4	Exemplo de configuração para ligação POP3 com ScalSec, no Thunderbird.	36
6.1	Mensagens relevantes na autenticação na nova solução.	42
6.2	Mensagens relevantes na autenticação no modelo PKIX.	43

Lista de Tabelas

6.1	Tempos entre pedido e resposta do DNS de um registo A.	43
6.2	Tempos de processamento do cliente, entre obtenção dos certificados e os pedidos LWRES ou OCSP, consoante se trata de ScalSec ou PKIX, respetivamente.	44
6.3	Tempos de processamento entre resposta aos pedidos LWRES ou OCSP e final da validação por parte do cliente.	44
6.4	Total dos tempos de processamento na validação do lado do cliente.	45
6.5	Número de certificados enviados para o cliente TLS no modelo PKIX.	46
6.6	Tamanho em bytes dos certificados PKIX transmitidos	47
6.7	Tamanho das chaves e assinaturas presentes nos certificados analisados.	48
6.8	Tamanho dos certificados excluindo as chaves e assinaturas.	49
6.9	Tamanho dos certificados considerando os tamanhos recomendados para as chaves.	49

Abreviaturas

ACL	Access Control List.
AD	Authenticated Data.
BIND	Berkely Internet Name Domain.
CA	Certification Authority.
CERT	Certificate.
CNAME	Canonical Name.
CPS	Certification Practice Statement.
CRL	Certificate Revocation List.
CRT	Certification Revocation Tree.
DHCP	Dynamic Host Configuration Protocol.
DIG	Domain Information Groper.
DN	Distinguished Name.
DNS	Domain Name System.
DNSKEY	Domain Name System Key.
DNSSec	Domain Name System Security Extensions.
DoS	Denial of Service.
DS	Delegation Signer.
FQDN	Fully Qualified Domain Name.
HTTP	Hypertext Transfer Protocol.
HTTPS	HTTP over SSL/TLS.
IETF	Internet Engineering Task Force.
IKS	Internet Key Service.
IMAP	Internet Message Access Protocol.
IMAPS	Internet Message Access Protocol Secure.
IP	Internet Protocol.
IPSEC	Internet Protocol Security.
KEY	Application Key.
KSK	Key Signing Key.
LWRES	Lightweight Resolver Library.
LWRESD	Lightweight Resolver Library Daemon.

MAC	Message Authentication Code.
MITM	Man-in-The-Middle.
MX	Mail Exchanger.
NIST	National Institute of Standards and Technology.
NSPR	Netscape Portable Runtime.
NSS	Network Security Services.
OCSP	Online Certificate Status Protocol.
PEM	Privacy Enhanced Mail.
PGP	Pretty Good Privacy.
PKI	Public-Key Infrastructure.
PKIX	Public-Key Infrastructure (X.509).
POP3	Post Office Protocol.
POP3S	Post Office Protocol Secure.
PSM	Personal Security Manager.
RR	Resource Record.
RRSIG	Resource Record Signature.
SDSI	Simple Distributed Security Infrastructure.
SMTP	Simple Mail Transfer Protocol.
SPKI	Simple Public Key Infrastructure.
SSH	Secure Shell.
SSHFP	Secure Shell Fingerprint.
SSL	Secure Sockets Layer.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
TSIG	Transaction Signature.
TTL	Time to live.
TTP	Trusted Third Party.
UDP	User Datagram Protocol.
URL	Uniform Resource Locator.
ZSK	Zone Signing Key.

Capítulo 1

Introdução

1.1 Enquadramento

A evolução da *Internet* permitiu o desenvolvimento das comunicações e de mercados de negócio *online*, mas apresenta problemas de segurança. O principal problema é o facto de se tratar de uma rede de comunicação aberta onde é possível permanecer no anonimato. Naturalmente, surgem problemas ao nível da identificação das partes envolvidas numa comunicação. A informação pode ser captada, alterada por terceiros, ou ser completamente perdida. A negação de envolvimento numa transacção de informação (repúdio) é outro problema. O acesso e controlo de recursos tornam-se muitas vezes um problema associado.

Os principais serviços são disponibilizados sobre um meio por natureza inseguro. O [DNS \(*Domain Name System*\)](#) é um elemento fundamental e crítico no suporte da *Internet*. Ataques como o [MITM \(*Man-in-The-Middle*\)](#) ou falhas de [DoS \(*Denial of Service*\)](#) são possíveis.

Estas e outras questões têm de ser resolvidas para garantir a segurança das comunicações. Dependendo dos serviços ou aplicações, podem existir várias soluções que tentam garantir comunicações seguras.

Os serviços mais importantes são, essencialmente, os disponibilizados por servidores [HTTP \(*Hypertext Transfer Protocol*\)](#). [IMAP \(*Internet Message Access Protocol*\)](#), [POP3 \(*Post Office Protocol*\)](#) e [SMTP \(*Simple Mail Transfer Protocol*\)](#) são igualmente exemplos de serviços importantes. O [SSL \(*Secure Sockets Layer*\)](#) foi desenvolvido pela *Netscape*, para fornecer segurança ao serviço [HTTP](#). O [HTTPS \(*HTTP over SSL/TLS*\)](#) consiste no [HTTP](#) sobre [SSL/TLS \(*Transport Layer Security*\)](#). O [SSL](#) permite evitar a escuta dos dados transmitidos e ataques [MITM](#). [SSL](#) e [TLS](#) foram estendidos para os outros serviços. O protocolo [TLS](#) é uma norma do [IETF \(*Internet Engineering Task Force*\)](#) baseada, mas não compatível com o [SSL](#).

O [SSL/TLS](#) depende do [PKIX \(*Public-Key Infrastructure \(X.509\)*\)](#) para a autenticação e distribuição de chaves. O [PKIX](#) é o perfil do X.509 para a *Internet*. No caso de servidores *Web*, o [SSL/TLS](#) é transparente para o utilizador, um certificado X.509 é

enviado para o cliente para autenticar o servidor. O cliente/utilizador também pode ser autenticado recorrendo a um certificado, contudo, tipicamente, apenas o servidor é autenticado. Não existe um mecanismo global para implementar massivamente certificados de clientes. Contudo é possível realizar este tipo de ataques sob determinadas condições [1].

Apesar do avanço da implementação global do [DNSSec \(Domain Name System Security Extensions\)](#) ser alvo de relativo impedimento, o mesmo deverá ser implementado no futuro para garantir a segurança do [DNS](#). Essa implementação pode ser aliada à disponibilização de chaves para [IPSEC \(Internet Protocol Security\)](#) ou outras aplicações.

O presente trabalho propõe uma implementação de um modelo de certificação distribuída para autenticação de sessões [SSL/TLS](#) recorrendo ao [DNSSec](#). Para isso é adicionado um novo registo ao [DNSSec](#). Pretende-se que a implementação seja transparente para o utilizador.

O cliente é melhorado para suportar o novo modelo em paralelo com o já existente ([PKIX](#)). Neste trabalho a implementação ao nível do cliente foi realizada no código do projecto *Mozilla*, o qual é comum a várias aplicações, salientando o *Firefox* e o *Thunderbird*. Estas aplicações implementam alguns dos principais serviços a disponibilizar ao utilizador, nomeadamente, a *Web* e o e-mail.

1.2 Modelo de certificação ScalSec

O presente trabalho de implementação é baseado num novo modelo de certificação para sessões [TLS](#), que constitui uma parte de um modelo mais abrangente, o ScalSec. O novo modelo é o resultado dos trabalhos de doutoramento do co-orientador desta tese, Eng. Jaime Dias, colaborador da UTM (Unidade de Telecomunicações e Multimédia) do INESC Porto (Instituto de Engenharia de Sistemas e Computadores do Porto).

O ScalSec é caracterizado por uma arquitectura hierárquica e distribuída, seguindo a estrutura administrativa, onde o [DNSSec](#) assina e distribui o resumo de uma chave por nó. O nó por sua vez assina uma chave por aplicação (serviço). Tanto o nó como o serviço recorrem a certificados X.509. O novo modelo é orientado para a *Internet*, para um espaço de nomes ao nível global. Até ao nó a confiança é estabelecida pelas entidades que gerem o domínio de nomes ao qual o nó pertence. Do nó para o serviço, a confiança é estabelecida pela entidade que gere o nó (o seu administrador). Apenas a [CA \(Certification Authority\)](#) local, ao nível do nó, emite os certificados para os respectivos serviços, cujos identificadores são o par protocolo de transporte/porto onde o serviço está à escuta no nó.

Na figura 1.1 pode-se observar a estrutura do modelo ScalSec.

1.3 Arquitectura da implementação

A solução implementada divide-se em três partes fundamentais:

- Adição ao [DNSSec](#) de novos registos com o resumo de chave pública por nó.

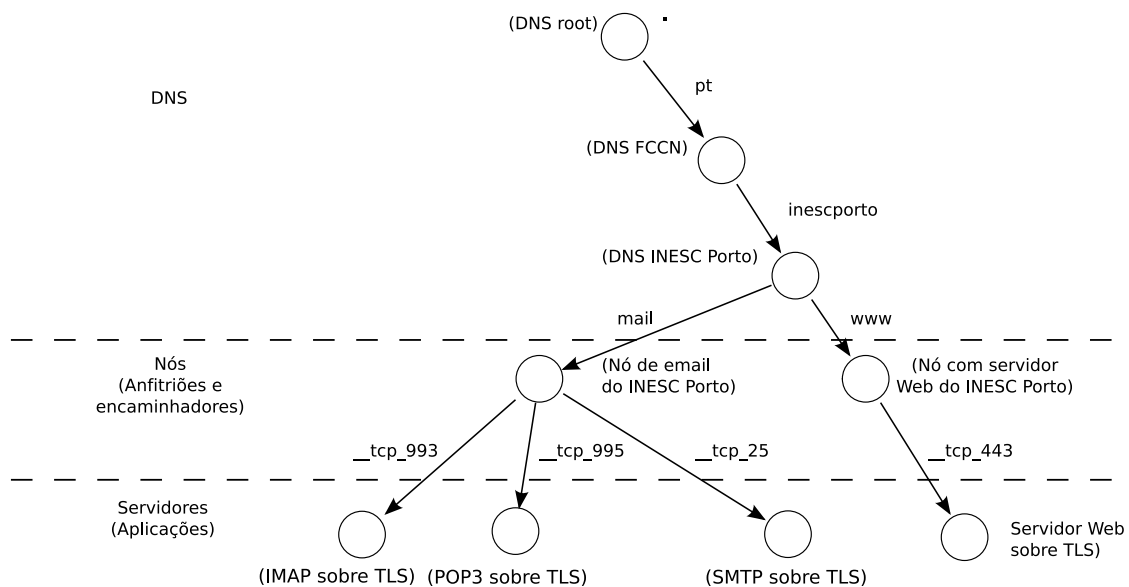


Figura 1.1: Modelo de certificação ScalSec.

- Emissão de certificados X.509 melhorados, por serviço, cuja **CA** é o nó local.
- Alteração de um cliente **SSL/TLS**, nomeadamente o *Mozilla* para fazer a validação da cadeia de certificados, recorrendo ao **DNSSec**.

Na figura 1.2 pode-se observar as principais relações entre as partes fundamentais da solução implementada. Inicialmente foi configurado o **DNSSec**, para posteriormente adicionar os registos com o resumo da chave do nó. Seguidamente, é dada atenção à forma como os certificados são gerados no próprio nó. Os serviços são configurados com os respectivos certificados, e o resumo da chave, que representa o nó, obtido e adicionado ao **DNSSec**. Por último, a validação dos certificados, recorrendo ao **DNSSec** segundo o modelo ScalSec, é implementada. Seguidamente, apresenta-se a descrição de cada uma das partes da implementação.

1.3.1 DNSSec

O ScalSec depende da configuração do **DNSSec**. Os servidores têm de ser configurados para utilizar as extensões de segurança. O resumo de uma chave pública, associada ao nó, é colocado no **DNSSec** através de um registo definido para o efeito. O registo foi definido como sendo do tipo H, e está associado ao **FQDN** (*Fully Qualified Domain Name*) do nó. O registo H possui, para além do resumo da chave pública, a informação acerca dos algoritmos de chave e resumo. O **DNSSec** fornece a autenticidade e integridade na obtenção do registo associado ao **FQDN**, independentemente de o mesmo ser pedido pelo próprio **FQDN** ou por um nome sinónimo, um **CNAME** (*Canonical Name*).

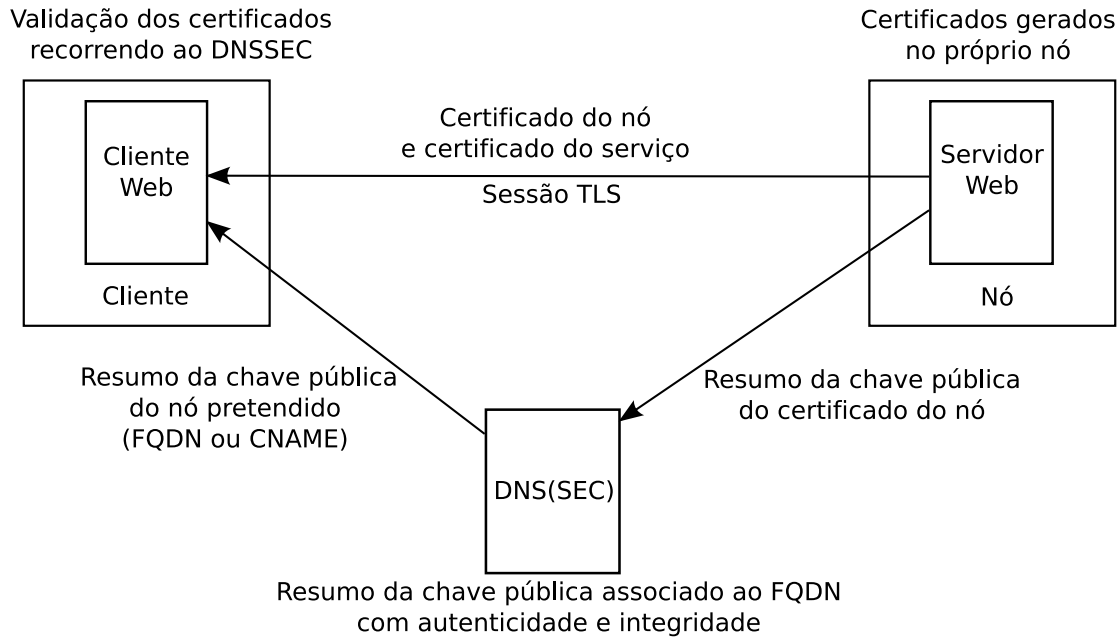


Figura 1.2: Arquitectura geral da implementação do novo modelo.

1.3.2 Servidores

No modelo ScalSec os servidores são certificados pelo próprio nó. Para possibilitar ligações seguras aos clientes não é necessário alterar o modo como funciona o serviço ou aplicação, apenas os certificados utilizados para disponibilizar as ligações seguras são alvo de alteração. Para cada nó é necessário criar uma [CA](#) local associada ao nó.

A chave pública do certificado da [CA](#) local, que é criada para o nó, é utilizada para representar o nó. O resumo da chave pública é colocado num registo do tipo H associado ao [FQDN](#) do nó. O certificado da [CA](#) local possui um identificador que permite ao cliente [TLS](#) saber qual o modelo de certificação utilizado, [PKIX](#) ou ScalSec. Os certificados são gerados recorrendo a ferramentas X.509 convencionais, nomeadamente o *OpenSSL* [2]. Cada serviço tem um certificado, o qual contém o par de identificadores pelo qual o serviço é identificado na pilha [TCP \(Transmission Control Protocol\)/IP \(Internet Protocol\)](#), nomeadamente, o par protocolo/porto. São utilizados certificados de curta duração. Deste modo, no caso das chaves privadas ficarem comprometidas, ou os dados no certificado já não estarem correctos, é suficiente aguardar pela expiração do certificado, não sendo necessário o recurso a mecanismos de revogação explícitos como acontece, tipicamente, no [PKIX](#).

1.3.3 Cliente

No servidor são disponibilizados os certificados do nó e dos serviços. O cliente verifica se os certificados são válidos no momento e para o serviço a que se referem. O resumo

da chave pública do nó é comparado com o resumo presente num registo [DNS](#) do tipo H associado ao [FQDN](#) do nó. A informação presente no [DNSSec](#) é verificada seguindo e verificando as assinaturas presentes nos registos [RRSIG](#) (*Resource Record Signature*), até que seja encontrada uma chave de confiança. Idealmente, deveria haver apenas uma chave de confiança, a da raiz [DNS](#), actualmente inexistente. O cliente [TLS](#) é melhorado para suportar a nova solução, mantendo a compatibilidade com o [PKIX](#). A autenticação segundo o novo modelo é realizada sempre que os certificados indiquem a organização 'O=ScalSec' no [DN](#) (*Distinguished Name*). O processo de verificação dos certificados X.509 é igual ao convencional, com excepção da verificação do identificador do serviço, o qual tem de coincidir com o protocolo/porto usado. A validação da [CA](#) é feita recorrendo aos registos tipo H acrescentados ao [DNSSec](#). A solução permite o suporte seguro de sinónimos de nomes de domínio (registos [CNAME](#)) através do [DNSSec](#).

1.4 Objectivos e contribuições relevantes

Os objectivos desta dissertação são os seguintes:

- Implementação de extensões para o *Mozilla NSS* (*Network Security Services*) e o [DNSSec](#) que permitam a autenticação de servidores [TLS](#) de forma segura e transparente para o utilizador, segundo o modelo de certificação ScalSec.
- Validação da solução em termos de funcionalidade, interoperabilidade e desempenho.

As contribuições relevantes do trabalho desenvolvido são as seguintes:

- O cliente [TLS](#), parte integrante do *Mozilla NSS*, é alterado de modo a suportar o ScalSec em paralelo com o [PKIX](#). O cliente é modificado para recorrer a certificados X.509 adaptados e a novos registos por nó, presentes no [DNSSec](#), para autenticar o servidor [TLS](#), suportando sinónimos ([DNS](#)) de forma transparente e segura, sem modificações dos referidos certificados.
- Define-se um novo registo [DNSSec](#) para suporte de resumos de chaves públicas associadas a nomes ou endereços de nós, *hosts* ou *routers*.
- A semântica dos certificados X.509 é adaptada de modo a reflectir o novo caminho de certificação, bem como os novos identificadores. Cada nó assume-se como [CA](#) local e emite certificados para cada serviço.
- São implementados mecanismos ao nível do nó servidor, que permitem a gestão automática dos respectivos certificados, bem como a actualização dos respectivos registos no [DNS](#), de forma transparente para o utilizador e sem alteração dos processos servidores.

1.5 Estrutura da tese

A tese está estruturada de acordo com o seguinte:

- O capítulo 1 apresenta o enquadramento do problema a resolver, a arquitectura da implementação, objectivos, contribuições relevantes e a estrutura deste relatório.
- O capítulo 2 refere o estado da arte no âmbito do trabalho desenvolvido.
- O capítulo 3 descreve o desenvolvimento no que respeita ao [DNSSec](#).
- O capítulo 4 descreve o desenvolvimento do modelo do ponto de vista dos servidores.
- O capítulo 5 descreve o desenvolvimento do modelo do ponto de vista do cliente.
- O capítulo 6 realiza uma análise de resultados sobre o trabalho desenvolvido.
- O capítulo 7 realiza uma revisão do trabalho desenvolvido. São referidas as conclusões e possibilidades de trabalho futuro a desenvolver.
- O anexo A possui os aspectos relativos a configuração do [DNSSec](#).
- O anexo B possui exemplos e aspectos relativos à configuração dos servidores.
- O anexo C possui as configurações e código necessário para o cliente.

Capítulo 2

Estado da arte

Neste capítulo descreve-se o estado da arte associado ao [SSL/TLS](#), bem como as infraestruturas para a autenticação das entidades envolvidas. São referidas as vantagens e desvantagens do [PKIX](#), do qual depende o [SSL/TLS](#). Mencionam-se as características essenciais do [PGP](#) (*Pretty Good Privacy*) e do [SPKI](#) (*Simple Public Key Infrastructure*)/[SDSI](#) (*Simple Distributed Security Infrastructure*). Referem-se ainda as vantagens e desvantagens na implementação do [DNSSec](#) e especificações no que diz respeito a chaves de aplicação e certificados.

2.1 SSL/TLS

O protocolo [SSL](#) foi desenvolvido pela *Netscape* para fornecer serviços de segurança a clientes e servidores *Web*. O protocolo [TLS](#) é o sucessor do [SSL](#), mas não é compatível com este. O protocolo [TLS](#) é constituído por camadas, responsáveis pelo estabelecimento, mensagens de alerta, finalização de estabelecimento, e pelo transporte de informação de aplicação. Durante a negociação de estabelecimento, o cliente e servidor determinam os mecanismos criptográficos necessários.

Tipicamente, a negociação inicia-se com o pedido de ligação segura ao servidor, por parte do cliente com sugestões para os algoritmos a usar no estabelecimento da ligação. O servidor responde com os algoritmos escolhidos ou identificador de uma sessão, para restabelecer uma sessão já negociada anteriormente. Seguidamente, o servidor envia os certificados para se identificar perante o cliente. O cliente verifica os certificados, podendo solicitar informação de terceiros. Após a validação o cliente envia uma chave para o servidor. Esta chave permite gerar as chaves para cifrar e autenticar as comunicações das aplicações.

O [SSL/TLS](#) por si só não realiza a autenticação das entidades, dependendo externamente do [PKIX](#) para tal. A confiança da [CA](#) depende apenas do facto de estar presente na lista das [CA](#) do browser. A validade resulta da verificação das assinaturas e períodos de validade. Tipicamente, não se verifica a revogação de certificados. O nome presente no

certificado tem de corresponder ao nome que é utilizado para realizar a resolução através do [DNS](#).

Ao nível da confidencialidade refere-se que no [SSL](#) um *eavesdropper* pode ter acesso a informação não cifrada e conhecida de um modo geral, mas que o algoritmo de cifragem resiste a ataques [3].

A possibilidade de retomar sessões, já estabelecidas anteriormente, é uma vantagem na medida em que evita as operações de algoritmos de criptografia assimétrica, evitando sobrecarregar os servidores. Através do identificador da sessão pode-se reutilizar a chave simétrica dentro de um período de tempo.

Ao nível da autenticação, o [SSL/TLS](#), por si só, não garante que de facto se está a comunicar com a entidade que se pretende [3]. Para além disso, apesar de permitir a autenticação das entidades envolvidas na comunicação, normalmente, apenas os servidores são autenticados. Não existe um mecanismo para implementar massivamente a autenticação dos clientes. O problema não está no [SSL](#) em si, mas no facto de depender de [PKIX](#) para a autenticação. Em termos de segurança, este é o aspecto mais importante no [SSL/TLS](#). O [SSL/TLS](#) de um modo geral, e tomando as devidas considerações e precauções, é seguro.

2.2 PKIX

O [PKIX](#) é o perfil do X.509 para a *Internet*. [PKIX](#) é a solução de certificados dominante nas aplicações *Web* em geral possuindo um formato flexível. O modelo de certificação é baseado em [TTP](#) (*Trusted Third Party*), as [CA](#), que emitem certificados de acordo com políticas, as [CPS](#) (*Certification Practice Statement*), estabelecidas pelas mesmas [TTP](#). O modelo actual de certificação é caracterizado por [CA](#) comerciais que desempenham um papel de notário. As [CPS](#) descrevem os métodos seguidos para emitir e gerir os certificados. O modelo de confiança é caracterizado por [CA](#) organizadas em hierarquia. Actualmente a hierarquia é definida por várias [CA](#) comerciais de topo ou de raiz que possuem uma hierarquia interna, estritamente hierárquica. Não existe certificação cruzada entre [CA](#), cada [CA](#) de topo ou de raiz não tem qualquer relação com as outras.

Para verificação por parte das aplicações, estas possuem listas de certificados das referidas [CA](#) comerciais, que incorrem em custos para fazer parte das mesmas listas. Um utilizador por sua vez incorre frequentemente em custos económicos e administrativos para emissão de certificados pela [CA](#).

O esquema [CRL](#) (*Certificate Revocation List*) faz parte da norma X.509. Uma [CRL](#) é uma lista de números de série de certificados revogados de uma [CA](#), assinada digitalmente e marcada temporalmente com período de validade. [CRL](#) é o método tradicional para revogação de certificados. É emitida periodicamente pela [CA](#) para o domínio da mesma. Espera-se que quem verifica o certificado, verifique as [CRL](#) antes de usar o mesmo. O esquema [CRL](#) não é eficaz porque existe sempre atraso entre a revogação e a emissão da [CRL](#). Mesmo que a [CRL](#) seja frequentemente emitida isso implica por outro lado a

obtenção frequente por parte dos clientes para verificação segura. Uma [CRL](#) pode ser muito longa implicando a comunicação de informação desnecessária.

O [OCSP](#) (*Online Certificate Status Protocol*) é um protocolo normalizado para confirmação da revogação de certificados. É indicado para aplicações que dependem criticamente de informação de revogação recente. Uma sessão [OCSP](#) implica a troca de mensagens entre cliente e servidor com informação de revogação. O protocolo estabelece a forma como é enviado o pedido de informação acerca dos certificados, as respostas a dar e as mensagens de erro. As respostas são assinadas digitalmente quer seja pela [CA](#) ou entidade de reconhecida confiança. Implica elevado processamento que se agrava com vários pedidos de estado de certificados. O protocolo define três valores de estado diferentes para um certificado. As definições são muito vagas no que respeita a validade do certificado. O protocolo não define a forma como a informação é tratada no servidor ou cliente, apenas estabelece a troca de mensagens.

Uma [CA](#) comercial não possui autoridade nas atribuições de nomes de domínio nem no registo de nomes de organizações [4]. Quando muito tem autoridade em dar a permissão a um servidor para estabelecer uma conexão [SSL/TLS](#). As [CPS](#) são definidas pelas próprias [CA](#). Os métodos seguidos para emitir e gerir os certificados descritos pelas [CPS](#) são frequentemente alvo de crítica. As [CA](#) determinam a seu favor as questões legais e responsabilidades. O grupo de trabalho do [PKIX](#), no [IETF](#), é em grande parte formado pelas [CA](#) comerciais que influenciam o desenvolvimento dos trabalhos a seu favor, privilegiando o modelo de negócio.

As [CA](#) comerciais incorrem em custos para fazer parte das listas de certificados das aplicações. O utilizador por sua vez incorre frequentemente em custos económicos e administrativos para a emissão de certificados pela [CA](#). O lucro está implícito e depende muitas vezes do que está estabelecido nas [CPS](#).

Alguns servidores *Web* emitem os seus próprios certificados e outros usam certificados obtidos de [CA](#) que não verificam, exaustivamente, as identidades que certificam. A segurança na associação de uma identidade verdadeira a uma chave pública tem de ser garantida por uma política de identificação por parte de uma [CA](#). Para além da correcta identificação, é necessário garantir que a mesma identidade possui de facto a respectiva chave privada [4]. [CA](#) falsas sem confiança podem ser adicionadas às listas de certificados. As cadeias de confiança dos certificados são caracterizadas por inconsistências e estão quebradas.

Não se pode estabelecer o não repúdio quando os sistemas de armazenamento de chaves privadas não são seguros. O conceito de não repúdio na realidade não é um conceito linear como na teoria.

Refere-se frequentemente que o [DN](#), no papel de identificador global pode não ser único [4]. Confiar demasiado num [DN](#) pode resultar em estratégias de autenticação condenadas ao fracasso [5]. Contudo, no caso de servidores *Web*, os [DN](#) são nomes de [DNS](#), que são únicos no domínio do seu espaço de nomes [6], que é caracterizado por uma única

raiz. Neste caso o problema não está no [DN](#) presente no certificado mas sim nas questões relacionadas com a integridade e autenticidade do [DNS](#).

A informação do servidor presente no certificado, quer seja no [DN](#) ou *Subject Alternative Name*, tem de corresponder ao introduzido na aplicação, por exemplo: [URL \(Uniform Resource Locator\)](#) no caso do Web Browser. Sem o [DNSSec](#), o [DNS](#) não garante a sua informação com integridade e autenticidade, de modo que o uso dos registos [CNAME](#), para uma maior flexibilidade na gestão dos nomes, não é actualmente suportado durante o estabelecimento de sessões [SSL/TLS](#).

Não existe a certeza que se está a comunicar com a entidade pretendida. A maior parte dos utilizadores tem dificuldade em analisar a autenticidade, o papel dos indicadores da mesma e compreender os certificados ou a forma como os mesmos são apresentados [7]. O projecto das interfaces de utilizador não considera os factores e limitações humanas. Frequentemente, os ataques dependem da incapacidade de análise e das decisões incorrectas por parte dos utilizadores.

Os períodos de validade dos certificados em geral são longos. Para além dos custos de emissão um dos exemplos é o facto dos certificados expirarem num efeito dominó [8]. Se o certificado de uma [CA](#) expira, todos os certificados emitidos por aquela perdem a validade. A possibilidade de usar certificados com períodos de validade mais reduzida para reduzir os riscos seria desejável mas requer maior carga de trabalho por parte de uma [CA](#).

Tipicamente, não se verifica a revogação de certificados. As [CRLs](#) não são verificadas. Apenas pode existir uma [CRL](#) para o domínio de uma [CA](#) num determinado momento. As extensões Delta [CRL](#) e de pontos de distribuição [CRL](#) melhoram a eficiência das [CRL](#) mas adicionam maior complexidade. A eficiência da verificação de revogação pode ser melhorada com [CRT \(Certification Revocation Tree\)](#), que requer menor largura de banda que noutros casos. A actualização dos nós, que são dependentes implica a reconstrução de grande parte ou toda uma árvore. O [OCSP](#) permite obter informação mais actualizada, não sendo necessário transferir uma [CRL](#) completa. Ainda assim, o tráfego e processamento associados ao [OCSP](#) não são desprezáveis, o que leva as [CA](#) comerciais a manter períodos de revogação elevados (1 semana a 1 mês).

Não existe um mecanismo explícito de verificação de revogação de certificados óptimo. Cada um pode ser aplicado de acordo com os requisitos que se pretendem: período de actualização, segurança, desempenho em largura de banda ou processamento. É possível combinar os mecanismos de revogação com melhores resultados mas aumentando a complexidade da implementação podendo originar problemas de decisão na revogação dos certificados. Idealmente, seria desejável evitar usar qualquer mecanismo explícito de verificação de revogação [5].

2.3 PGP e SPKI/SDSI

O [PGP](#) possui a capacidade de fornecer serviços com confidencialidade e autenticação. É bastante usado em aplicações de correio electrónico e armazenamento de ficheiros. Cada utilizador possui o seu conjunto de pares de chaves privada e pública que podem ser mais do que um.

O modelo de confiança do [PGP](#) é a *Web of Trust*. A *Web of Trust* é um conceito que se define pelo estabelecimento de redes de confiança entre grupos de utilizadores. Os utilizadores estabelecem uma rede entre eles de acordo com a confiança que têm com quem comunicam, assinando os certificados uns dos outros. A confiança pode ser num certificado assinado por um utilizador que declara a autenticidade do mesmo, mas também pode ser num assinante para a certificação de outros certificados. A confiança é estabelecida em níveis de maior ou menor confiança de acordo com a opinião de cada utilizador. A confiança de um utilizador, num assinante, é mantida secreta e apenas tem significado para esse mesmo utilizador.

O [PGP](#) funciona relativamente bem para grupos pequenos de pessoas em ambiente controlado. Não permite cadeias de confiança. Adapta-se bem para aplicações tais como o correio electrónico mas é escalável ao ponto de garantir a segurança ao nível do comércio electrónico. Como se baseia no conceito de *Web of Trust*, a autenticação depende da confiança que outros utilizadores atribuem. Para o caso de servidores *Web* não é o mais indicado.

O [SPKI/SDSI](#) é a fusão de dois projectos. [SPKI](#) centra-se na simplicidade e flexibilidade das questões de autorização. [SDSI](#) possui uma arquitectura de espaço de nomes ao nível local para fornecer um serviço seguro e escalável. O [SPKI/SDSI](#) tem apenas significado local, mais orientado para a autorização e frequentemente associado a [ACL](#) (*Access Control List*). Existem trabalhos desenvolvidos [9] [10], utilizando [SPKI/SDSI](#), que demonstram isso mesmo. É claramente indicado para acesso a recursos ou permissões, permitindo delegação dos mesmos. Para autenticação num espaço de nomes, embora distribuído, mas global, como o [DNS](#) não é a opção mais indicada. Embora assente num modelo distribuído, o [SPKI/SDSI](#) trata os nomes [DNS](#) e endereços [IP](#) segundo a mesma abordagem do [PKIX](#), as [CA](#) passam a agir como notários.

2.4 DNSSec

O [DNS](#) é uma base de dados distribuída com o objectivo de resolução de nomes de domínio em endereços [IP](#) e vice-versa. A arquitectura é hierárquica e descentralizada. O [DNS](#) é vulnerável à corrupção de dados, poluição de cache ou personificação de um servidor por um forjado. O [DNSSec](#) é a concretização das extensões de segurança ao [DNS](#), corrigindo várias falhas do [DNS](#). Recorrendo a assinaturas dos registos das zonas por parte

dos servidores, através da adição de novos registos, os clientes podem verificar a validade das mesmas assinaturas. Pode-se assim verificar a autenticidade e integridade dos registos.

O [DNSSec](#) por si só implica custos acrescidos em relação ao [DNS](#). Refere-se por vezes que é economicamente ineficiente. Os custos de largura de banda e espaço de armazenamento sofrem aproximadamente um aumento seis vezes em relação ao [DNS](#) [11]. Até à data, e face às ameaças correntes, isoladamente não justifica o investimento na implementação. Contudo o suporte de distribuição de chaves como por exemplo para [IPSEC](#) ou [SSH \(Secure Shell\)](#) poderia adicionar valor acrescido para justificar esse mesmo investimento. O [DNS](#) já é caracterizado por seguir a estrutura administrativa da rede, o que o torna indicado para disponibilizar informação dos nós (*routers e hosts*).

A implementação dominante de [DNS](#), [BIND \(Berkely Internet Name Domain\)](#), já suporta o [DNSSec](#), disponibilizando ferramentas para a configuração e teste das extensões de segurança. Permite o uso de [TSIG \(Transaction Signature\)](#) para assinatura de pedidos realizados ao [DNS](#). Permite usar também [TSIG](#) para actualizações dinâmicas do [DNS](#) de forma segura. A familiarização com a implementação real do [DNSSec](#) torna-se em grande parte mais fácil nestas condições. O mesmo não pode ser dito em relação aos mecanismos de resolução fornecidos às aplicações. Existe, contudo, desenvolvimento e uma oferta variada em termos de ferramentas que permitem o desenvolvimento, implementação e manutenção do [DNSSec](#).

O [DNSSec](#) como repositório para distribuição segura de chaves é um concorrente das actuais [CA](#) comerciais.

Refere-se também que o [DNSSec](#) resolve a maior parte das ameaças de segurança [12] do [DNS](#). As ameaças que não resolve são sobretudo de carácter organizacional, que não podem ser resolvidas com *software*.

A implementação generalizada do [DNSSec](#) até à raiz do [DNS](#) ainda é alvo de alguma resistência. A implementação do [DNSSec](#) não se trata apenas de uma questão tecnológica, mas também de uma questão de interesses económicos [11]. Significa incorrer em custos económicos e administrativos na sua implementação e manutenção. Significa também o risco de perder negócios já estabelecidos onde já se investiu tempo e dinheiro.

Recorrendo ao [DNSSec](#) existem varias possibilidades de autenticação para serviços ou aplicações. Existem modelos para a disponibilização de chaves ou certificados por aplicação, serviço, ou utilizador. Adicionalmente, existem trabalhos que mantendo o modelo de [PKIX](#) recorrem ao [DNSSec](#) para garantir a integridade e autenticidade do [DNS](#), e especificar os requisitos de segurança de um servidor por meio de um registo [13]. Qualquer destas possibilidades requer a adição de pelo menos um registo ao [DNSSec](#).

2.4.1 Chaves de aplicação

O [DNSSec](#) permitiu a disponibilização de chaves de aplicação ou serviço [14]. Para cada aplicação ou serviço seria disponibilizada uma chave [15]. Um número elevado de serviços traduziria um número elevado de chaves presentes no [DNSSec](#), o que não é eficiente. O

uso de chaves de aplicação no DNSSec foi mais tarde limitado e revogado [16]. As razões para tal devem-se ao facto de sobrecarregar o DNSSec com imensos registos no caso de disponibilização de inúmeros serviços. Para além disso, existe o argumento de que o registo KEY (*Application Key*) RR (*Resource Record*) ao disponibilizar chaves para vários serviços ou aplicações, daria a origem a supostos registos derivados de KEY RR. Esse facto iria dar origem a problemas na arquitectura do DNSSec.

Existem desenvolvimentos que permitiriam disponibilizar as referidas chaves em servidores dedicados. Possibilidades são os casos de protocolos dedicados [17] como por exemplo o IKS (*Internet Key Service*) [18]. Implica custos para manter os servidores adicionais.

2.4.2 Certificados

A utilização do CERT (*Certificate*) RR no DNS [19] implica o funcionamento deste como repositório de certificados e, consequentemente, o aumento de tráfego. Permite um mecanismo de distribuição de certificados para uma PKI (*Public-Key Infrastructure*). Não implica um acréscimo em termos de segurança. Apenas uma maior disponibilidade.

Ao recorrer ao DNSSec para assinar os certificados, as limitações são semelhantes ao uso de chaves de aplicação através do KEY RR. Pelo seu tamanho, o tráfego é agravado com o uso de certificados em relação ao uso de chaves de aplicação, é ainda menos eficiente do que o uso de chaves por serviço, ou aplicação. A maior parte das implementações que recorrem a este tipo de registos normalmente implementam o DNSSec com recurso a registos de apontadores para outros servidores DNSSec especialmente orientados para funcionar apenas como repositórios de CERT RR. Implica maiores custos de implementação pelo facto de ser necessário disponibilizar servidores dedicados para este tipo de registos.

Capítulo 3

DNSSec

O [DNSSec](#) tem um papel fundamental na implementação do ScalSec. Foi utilizado para mapear de forma segura os nomes/endereços IP nos nós. Primeiro são apresentados os registos utilizados ([DNSKEY](#) (*Domain Name System Key*)) e criados (H) para este trabalho, depois são apresentadas as linhas directas para a correcta instalação e configuração do [DNSSec](#) no âmbito deste trabalho. Finalmente, recorre-se ao [TSIG](#), para colocação do resumo de chave pública dos nós no [DNSSec](#).

3.1 Adaptação do DNSSec

Inicialmente, quando a informação a colocar no [DNSSec](#) era a chave pública, usava-se o registo [DNSKEY](#). Quando se implementou uma primeira solução com o resumo da chave, usava-se o registo [DS](#) (*Delegation Signer*). A utilização destes registos, embora não seja recomendada, não vai de encontro à configuração das zonas. Trata-se de registos com nomes de domínio associados a nós, e não a zonas. Contudo verificou-se que embora fosse possível usar o registo [DS](#), a implementação do [DNSSec](#) pelo [BIND](#) não permitia a assinatura das zonas se o [DS](#) fosse colocado manualmente. Embora fosse possível colocar o [DS](#) associado ao nó através de actualização dinâmica, manualmente o [BIND](#) assumia um erro. Tal erro especificava que era necessário existir um servidor de nomes associado ao nome do [DS](#). Para contornar este facto, e porque seria desejável que o registo também fosse colocado manualmente, foi criado um registo definido especificamente para a implementação. O registo foi designado por registo H. Este registo foi definido de forma semelhante a um já existente, o registo [SSHFP](#) (*Secure Shell Fingerprint*) [20], mas que está orientado para outra utilização, nomeadamente para aplicações [SSH](#).

O registo H definido teria os seguintes seguintes campos para os dados:

- O algoritmo da chave pública da qual se obtém o resumo.
- O algoritmo que é usado para obter o resumo da chave.
- O resumo propriamente dito.

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| algoritmo | tipo resumo |                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
/                                                                 /
/                                                                 /
/                                                                 /
/                                                                 /
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

O algoritmo da chave pública presente no registo é definido através dos seguintes valores:

- 0 - Reserved
- 1 - RSA
- 2 - DSA
- 3 - ECDSA

O algoritmo do resumo presente no registo é definido através dos seguintes valores:

- 0 - Reserved
- 1 - SHA1
- 2 - SHA-256
- 3 - SHA-512

Um exemplo para o registo H demonstra-se a seguir. O nome corresponde ao nome associado ao nó. Deve existir um registo A com o mesmo nome de domínio. O [TTL](#) (*Time to live*) definido no campo seguinte deve ser igual ao [TTL](#) dos registos A e [RRSIG](#) associados ao mesmo nome de domínio. Para o registo, de entre os números identificadores possíveis, escolheu-se o número 120 (0x78). O primeiro octeto (01) corresponde ao algoritmo da chave pública. O segundo octeto corresponde ao algoritmo do resumo (01). No exemplo dado trata-se de uma chave RSA e um resumo obtido por SHA1. O resto do campo de dados corresponde ao resumo. Antes do campo de dados do registo deve ser indicado o tamanho do mesmo. Neste caso trata-se de 22 octetos de informação, um para o algoritmo da chave, outro para o tipo de resumo, e os restantes 20 para o resumo da chave propriamente dito.

```
w3.porto.pt 3600 IN TYPE120 \# 22 01011f6fe8211a1d1189dfc7463b3c78adaa
6f05423
```

O registo H associado ao nome, com o resumo da chave pública e assinado pelo [DNSSec](#) é análogo a um certificado. Contudo se o nome associado ou o endereço [IP](#) for modificado

por qualquer razão, não é necessário alterar qualquer certificado, desde que as chaves não se alterem. Ao nível do [DNSSec](#) este registo tem um periodo de validade definido pelo [RRSIG](#) associado. Ao remover o respectivo [RRSIG](#) ou em caso do prazo do mesmo expirar, a validade do registo H e dos certificados é implicitamente revogada.

No novo modelo é necessário verificar que o resumo da chave pública a colocar, no registo H, referente ao nome de domínio do nó, está associado à correspondente chave privada do nó. Torna-se necessário a definição de políticas para as práticas referidas. Contudo podem ser mais objectivas e mais simples do que as políticas definidas para [PKIX](#). Além disso como o [DNS](#) é uma hierarquia distribuída em muitos casos pode existir uma aproximação e identificação fácil por parte das entidades autoritárias dos nós e pelas autoritárias do sistema de nomes de domínios.

3.2 Configuração do DNSSec

Na implementação e teste foram utilizadas duas máquinas para serem utilizadas como servidores [DNS](#). Uma delas seria autoritária pelo domínio `pt`. A outra seria autoritária pelo domínio `porto.pt`. Trata-se de uma estrutura local de dois níveis de nomes de domínio, isolada da estrutura global. Apenas o servidor responsável pelo subdomínio realizava *forwarding* para o outro servidor. Foram configuradas também, de modo semelhante, as respectivas zonas reversas. No estudo e configuração do [DNS](#) e das extensões foram consultadas várias fontes, desde manuais de software [21], tutoriais [22], livros [23] e guias práticos de configuração e administração [24].

Na figura 3.1 pode-se observar a estrutura da cadeia de confiança das zonas implementadas.

Para a implementação foi adicionado um registo A para ser utilizado no acesso e teste dos servidores. Foi adicionado um registo [CNAME](#) para apontar para o referido registo A. Este último registo seria utilizado para testar a utilização do registo [CNAME](#) na solução. Foi ainda adicionado um registo [MX](#) (*Mail Exchanger*) à zona para usar na implementação e teste dos servidores e do cliente de correio electrónico. O registo [MX](#) estava associado ao nome do respectivo registo A considerado.

Para activar as extensões de segurança do [DNS](#), o ficheiro `named.conf.options` deve possuir obrigatoriamente a opção:

```
dnssec-enable yes;
```

Com os servidores de [DNS](#) configurados para as zonas que se pretendem, podem-se criar as chaves para proceder posteriormente à assinatura das zonas. Deve-se produzir duas chaves, uma [ZSK](#) (*Zone Signing Key*) e uma [KSK](#) (*Key Signing Key*), para cada zona. Uma [KSK](#) pode ser considerada com um ponto de entrada seguro na estrutura do [DNSSec](#). Considerou-se que, tipicamente, uma [ZSK](#) teria uma validade maior que uma [KSK](#). Uma [ZSK](#) teria uma validade de um mês e uma [KSK](#) uma validade de um ano. Ao

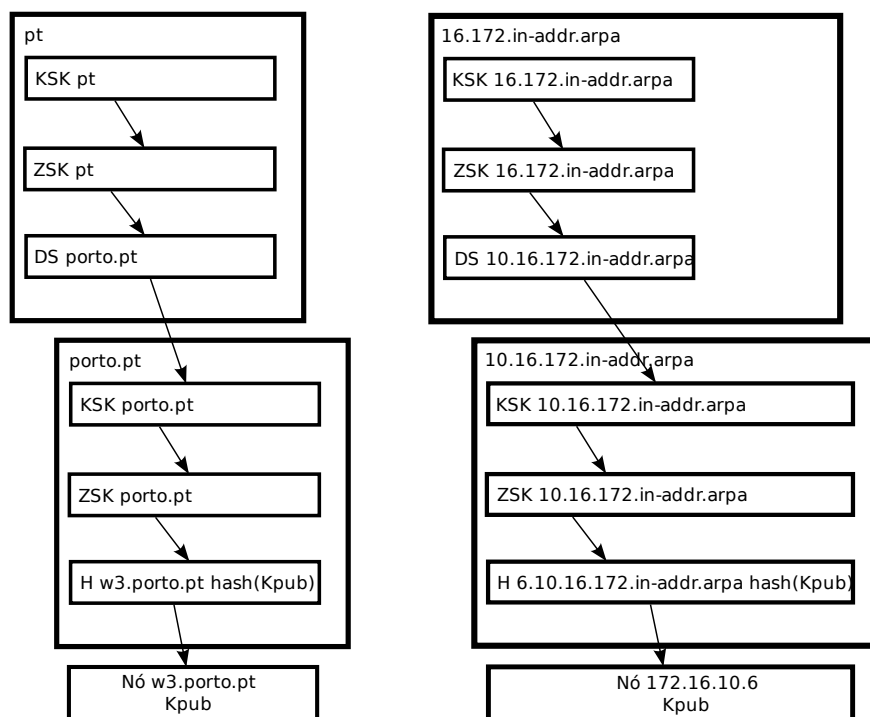


Figura 3.1: Estrutura da cadeia de confiança das zonas configuradas para a implementação do DNSSec.

longo da estrutura do [DNS](#), estas considerações podem ser igualmente alargadas. A raiz da estrutura deve ter chaves com maior tamanho que os seus subdomínios.

Para a implementação assumiu-se que o domínio `porto.pt` poderia ter uma [ZSK](#) de 1024 bits com a validade de um mês. A respectiva [KSK](#) poderia ser de 1300 bits com a validade de um ano. No domínio `pt` a [ZSK](#) seria de 1024 bits e a [KSK](#) de 1512 bits. Para a raiz da estrutura a [ZSK](#) seria de 1024 bits e a [KSK](#) de 4096 bits.

Para gerar as chaves e efectuar a respectiva assinatura das zonas utilizam-se as ferramentas disponíveis pelo [BIND](#). Para o domínio `porto.pt` as chaves poderiam ser geradas da seguinte forma:

```
$ dnssec-keygen -r/dev/urandom -a RSASHA1 -b 1024 -n ZONE porto.pt
$ dnssec-keygen -r/dev/urandom -f KSK -a RSASHA1 -b 1300 -n ZONE
porto.pt
```

Igualmente para o domínio `pt` as chaves poderiam ser geradas da seguinte forma:

```
$ dnssec-keygen -r/dev/urandom -a RSASHA1 -b 1024 -n ZONE pt
$ dnssec-keygen -r/dev/urandom -f KSK -a RSASHA1 -b 1512 -n ZONE pt
```

Paralelamente, para as zonas reversas as considerações são as mesmas. As chaves podem ser geradas da mesma forma. Obtém-se assim um par de ficheiros para cada uma

das chaves, um para a privada e outro para a pública. A chave pública deve ser incluída no ficheiro da zona. A chave privada é utilizada para assinar a zona. Apresenta-se, seguidamente, um exemplo no caso em que a [ZSK](#) está no ficheiro `Kporto.pt.+005+50744.private` e a [KSK](#) está no ficheiro `Kporto.pt+005+15852.private`.

```
$ dnssec-signzone -o porto.pt db.porto Kporto.pt.+005+50744.private -k
Kporto.pt+005+15852.private
```

Depois de assinar as zonas basta reiniciar o servidor de nomes de domínio. Ao assinar as zonas obtém-se igualmente o registo [DS](#). Este registo deve ser colocado na zona situada acima da zona que foi assinada, na estrutura do [DNS](#). Para testar os servidores [DNSSec](#) pode-se usar o [DIG \(Domain Information Groper\)](#) que permite fazer o seguimento da validade das assinaturas em cadeia até uma chave de confiança. Para ter acesso a essa opção o [DIG](#) tem de ser compilado com a variável de ambiente `STD_CDEFINES` definida com o valor `DDIG_SIGCHASE=1`.

Inicialmente, para permitir o funcionamento básico do [DNSSec](#), basta apenas criar as chaves e assinar as zonas. Posteriormente é necessário gerir e manter o funcionamento. A total gestão e manutenção actualmente é dificultada, por não existirem ferramentas apropriadas para tais funções.

Para que seja permitido aos clientes realizar pedidos autenticados, é necessário que esteja igualmente uma chave presente na configuração dos servidores [DNSSec](#). Apresenta-se a chave para o caso da configuração implementada nos servidores de nomes.

```
key valter.porto.pt. {
    algorithm hmac-md5;
    secret "Me9gkynH0t1JViC6EfFpNA==";
};
```

O cliente assina os pedidos realizados ao servidor com a referida chave. No servidor de nomes, é suficiente existir a referida chave para que os pedidos assinados pela mesma sejam respondidos assinados pela mesma chave. Os nomes associados à chave têm de ser iguais no cliente e no servidor. O nome associado à chave será normalmente o nome de domínio da máquina cliente. Por exemplo, no caso em que os clientes acedem a uma rede recorrendo a um servidor [DHCP \(Dynamic Host Configuration Protocol\)](#), este mesmo poderia gerir a atribuição e associação das chaves aos nomes de domínio dos clientes. Significa uma chave adicional para a actualização dinâmica por parte do servidor [DHCP](#).

3.3 Actualização dinâmica do DNSSec

Para permitir a actualização de registos, podem ser configuradas as políticas de actualização para as zonas. É necessário criar uma chave para permitir a actualização de forma segura. A referida chave deve estar associada ao nome de domínio do respectivo nó,

e deve ser incluída nas configurações do servidor de nomes. De igual modo, como se trata de uma chave simétrica, quem realiza a actualização deverá possuir a mesma chave.

```
key w3.porto.pt. {  
    algorithm HMAC-MD5;  
    secret "ASP7f6GFZWqP+RAZFYTCGk0bZnyNIdmK0oW7QZ+ibkGdAHGp/MDh6YwB  
          m5/g8Axc+vHFZBatU5k2NSxgUPd10w==";  
};
```

É boa prática colocar as chaves num ficheiro dedicado e incluir o ficheiro na configuração de nomes do servidor de nomes. Isto faz-se não só por questões de segurança, mas também por questões de boas práticas de gestão das configurações dos servidores. Evitam-se ficheiros de configuração de difícil análise em casos onde existem inúmeras zonas. Apresenta-se o exemplo da configuração das mesmas políticas para o caso da implementação.

```
update-policy {  
    grant w3.porto.pt. name w3.porto.pt. ANY;  
};
```

A anterior configuração deve ser incluída no interior da definição da respectiva zona, no ficheiro de configuração do servidor de nomes. Permite actualizar todos os registos associados ao respectivo nome de domínio.

3.4 Conclusões

Foram configuradas as zonas para o [DNS](#), implementando-se as extensões de segurança. As zonas reversas foram igualmente implementadas de modo semelhante. Estudaram-se as possibilidades e limitações do [DNSSec](#). O registo H foi definido para permitir a associação a um nó de uma única chave. O [TSIG](#) foi utilizado para a configuração de actualizações dinâmicas e para a transacção segura entre cliente e servidor [DNS](#).

Capítulo 4

Servidores

Para gerar os certificados X.509 alterados por serviço é necessário criar os directórios de trabalho de uma CA local que representa o nó. O *OpenSSL* disponibiliza inúmeras opções de comandos de acordo com a documentação disponível, e permite fazê-lo de forma relativamente fácil e flexível. Existe documentação com exemplos para gerar CA locais que são em muito semelhantes ao que se pretende.

Para a implementação do servidor *Web* recorreu-se ao *Apache*. Para a implementação do servidor *SMTP* recorreu-se ao *Postfix*. Para a implementação dos servidores *POP3* e *IMAP* recorreu-se ao *Courier*.

Inicialmente criam-se os directórios necessários e uma base de dados para a gestão dos certificados gerados posteriormente.

4.1 A CA que representa o nó

Os directórios da CA são organizados da seguinte forma:

- **ScalSec**: contém o certificado da CA, certificados assinados pela CA, chaves privadas dos certificados, pedidos de certificação dos certificados, base de dados e ficheiros de configuração.
- **ScalSec/signedcerts**: contém as cópias de cada certificado assinado.
- **ScalSec/private**: contém a chave privada da CA.

Para gerar o certificado da CA e respectiva chave privada recorre-se a um ficheiro de configuração. Alguns argumentos fornecidos ao *OpenSSL* na linha de comandos prevalecem sobre as opções indicadas no ficheiro de configuração. O campo do nome da organização do certificado deve ser preenchido com 'ScalSec'. Por razões de compatibilidade com o *PKIX*, uma vez que a nova solução foi implementada paralelamente à já existente, o *DN* de cada certificado de nó (CA) tem de ser diferente de todas as outras; para isso, o 'commonName' é preenchido com a data da criação do certificado.

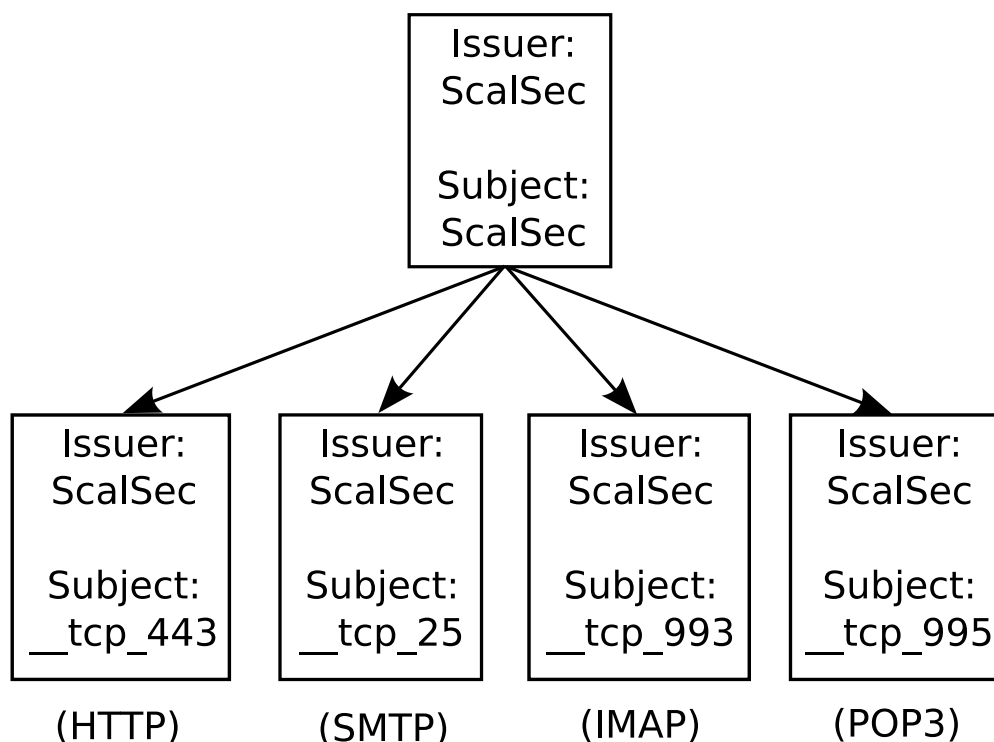


Figura 4.1: Estrutura da CA do nó e respectivos serviços.

```

commonName      = 2008-06-03 20:57:00.649319998+00:00
organizationName = ScalSec
  
```

O ficheiro deve declarar as seguintes extensões para assinalar o certificado do nó como uma [CA](#):

```

basicConstraints = CA:true, pathlen:0
nsCertType      = sslCA
  
```

Adicionalmente, assumiu-se um período de validade de 30 dias para o certificado da [CA](#) e um tamanho para a chave de 1024 bits.

```

$ openssl req -x509 -nodes -newkey rsa:1024 -out ScalSecCrt.pem -config
ScalSecConfig.cnf -days 30
  
```

A chave pública do certificado da [CA](#) local pode ser obtida recorrendo também ao *OpenSSL*. Através do mesmo é possível obter o resumo da mesma chave pública. Para obter um resumo SHA1 com uma chave RSA pode-se fazer do seguinte modo:

```

$ openssl x509 -sha1 -in ScalSecCrt.pem -noout -pubkey | openssl rsa
-pubin -outform DER | openssl dgst -sha1
  
```

A chave pública é obtida e convertida para formato binário. O resumo em formato hexadecimal obtém-se a partir da chave em formato binário. O resumo pode ser colocado num registo tipo H juntamente com informação acerca do algoritmo da chave e do tipo de resumo. Actualmente, apenas SHA1 é obrigatório pelo [DNSSec](#).

Independentemente do registo ser gerado manualmente ou recorrendo a um *script*, a chave pública deve ficar associada apenas ao [FQDN](#) do registo de tipo A. Para o suporte de nomes sinónimos ([CNAME](#)) a obtenção da chave pública pelo cliente é realizada de forma transparente. Na resolução de nomes de domínio basta que seja colocada apenas um registo no [DNSSec](#). Caso o serviço seja acedido através do endereço [IP](#) directamente, o resumo da chave deve ser colocado também associado à respectiva zona inversa.

4.2 Serviços

Para gerar um certificado por serviço é necessário criar um ficheiro de configuração com os dados necessários para os campos do sujeito do certificado. O campo da organização é preenchido com 'ScalSec' e o nome comum é preenchido com o par protocolo de transporte/porto, onde o servidor fica à escuta. Para o serviço [HTTPS](#), no porto 443, os campos são definidos no ficheiro de configuração da forma que se segue.

```
commonName      = __tcp_443
organizationName = ScalSec
```

Para evitar problemas de segurança em clientes [PKIX](#), é necessário restringir a política de utilização dos certificados de serviço, neste caso para servidor [SSL/TLS](#), e marcá-lo como não sendo uma [CA](#). O ficheiro de configuração da [CA](#) define para que utilização se destinam os certificados que são assinados pela mesma.

```
basicConstraints = CA:false
nsCertType      = server
extendedKeyUsage = serverAuth
```

Depois de editar o ficheiro de configuração para o serviço desejado é relativamente fácil gerar os certificados. Inicialmente, cria-se a chave com o tamanho que se pretende. Cria-se também o pedido que é usado para criar o certificado de serviço assinado pela [CA](#). Posteriormente, este é assinado com o período de validade especificado. A assinatura é realizada de acordo com a política e extensões definidas pela configuração da [CA](#). O período de validade e o tamanho da chave podem ser especificados ao nível do nó, de acordo com os serviços a disponibilizar. Para um serviço a disponibilizar permanentemente, poderia assumir-se um período de validade de 24 horas, com um tamanho de chave de 768 bits. De qualquer modo o período de validade de um certificado de serviço pode variar desde alguns minutos até ao máximo do período de validade do certificado da [CA](#).

```
$ openssl req -nodes -newkey rsa:768 -keyout __tcp_443_key.pem
-out __tcp_443_req.pem -config __tcp_443.cnf
$ openssl ca -in __tcp_443_req.pem -out __tcp_443_cert.pem -notext
-config ScalSecConfig.cnf -days 1
```

O certificado gerado em formato [PEM \(Privacy Enhanced Mail\)](#), em Base64, e a respectiva chave privada estão prontos a ser utilizados por exemplo pelo servidor *Web* ou [POP3](#). O certificado da [CA](#) que representa o nó deve ser adicionado também à configuração do serviço.

O custo de obtenção de certificados segundo o novo modelo é semelhante à emissão de certificados *self-signed*, permite maior controlo, por parte de quem gere o nó, na certificação dos respectivos serviços, em relação ao modelo [PKIX](#).

4.3 Optimização e automatização

Como se viu anteriormente, é relativamente fácil gerar manualmente a [CA](#) do nó e respectivos certificados para os serviços. Idealmente, isto é suficiente para teste ou casos isolados. Na prática é desejável uma abstracção dos certificados para quem realiza a administração do nó e dos respectivos serviços. Para além deste facto, os períodos de validade reduzidos em relação ao modelo [PKIX](#) significam uma maior frequência de renovação de certificados. Implica uma maior necessidade de gestão dos certificados e dos registos presentes no [DNSSec](#). Devido à cache dos registos do [DNSSec](#), é necessário a sobreposição temporal dos registos, no caso de substituição das chaves dos certificados da [CA](#) do nó.

A optimização e automatização do lado dos servidores foi implementada por um conjunto composto por *shell scripts*, um ficheiro de configuração e um ficheiro com informação para gerar os certificados de serviço ou aplicação. Foi implementada uma lógica de substituição de chaves, certificados e registos. Pela facilidade de emissão de novas chaves e certificados, não se considerou a funcionalidade de reutilização de chaves para diferentes certificados. Contudo essa funcionalidade pode ser adicionada futuramente. Se a chave pública em conjunto com a designação do sujeito da [CA](#) se mantiver, os certificados dos serviços não necessitam de renovação.

O *script* principal, *cacert*, gera os directórios da [CA](#), cria os ficheiros de configuração, e base de dados para informação acerca dos certificados assinados. O *cacert* é ainda responsável pela emissão dos certificados da [CA](#) e dos serviços ou aplicações, nos momentos de substituição dos certificados da [CA](#). Por questões de simplicidade, também foi desenvolvido um *script* secundário, *servicecert*. Este *script* realiza apenas a emissão e renovação dos certificados de serviço de forma independente do *cacert*. É permitida uma gestão apenas ao nível dos certificados dos serviços para possibilitar a gestão dos mesmos de modo flexível. As funções são semelhantes às utilizadas para os certificados de serviços no *cacert*.

As subsecções seguintes referem as considerações acerca das variáveis a considerar. Referem-se em particular as variáveis de estado para a implementação dos *scripts*. Outros

aspectos importantes são a actualização dinâmica dos registos no [DNSSec](#) e a gestão temporal de todo o processo.

4.3.1 Variáveis a considerar

O ficheiro de configuração especifica os valores das variáveis e opções a passar aos *scripts*. Permite passar os parâmetros necessários aos *scripts*. Pode ser estendido com novos argumentos. Para algumas variáveis são considerados valores por omissão.

Ao gerar os certificados manualmente, pode-se observar que as primeiras variáveis a considerar são os tamanhos das chaves e período de validade dos certificados. Para estas variáveis são especificados valores diferentes para a [CA](#) e os certificados dos serviços. Como já foi mencionado antes, os períodos de validade dos certificados podem ser relativamente curtos. Com prazos curtos de validade as chaves podem ser menores do que no caso [PKIX](#). Assumem-se como valores normais, os já mencionados, um prazo de um mês para [CA](#) e um período de 24 horas para os serviços. Para o tamanho das chaves, por omissão, são considerados os tamanhos de 1024 bits para a [CA](#) e 768 bits para os serviços.

Outro parâmetro que se revela fundamental é o [TTL](#) do registo H. Este parâmetro em conjunto com o período de validade associado ao certificado da [CA](#) é extremamente importante. Os dois valores associados permitem determinar os momentos de renovação e actualização de registos e certificados. São ainda especificadas algumas variáveis menos relevantes, que funcionam na decisão e escolha de opções.

4.3.2 Variáveis de estado

Devido ao facto dos registos [DNS](#) serem alvo de cache, é necessária sobreposição temporal na troca dos registos e esse aspecto revela-se na gestão da [CA](#). É sempre necessário colocar, no caso de troca de chave do certificado da [CA](#), o registo H com antecedência de pelo menos um [TTL](#) antes da substituição dos certificados nos respectivos serviços. Como se faz a troca de chave antes do final do período de validade do certificado da [CA](#), e a troca de chave revela-se na existência dos respectivos certificados, esta mesma foi escolhida como variável de estado para o *script* principal. Foram definidos três estados fundamentais, um estado de inicialização, um de início de sobreposição temporal e um de fim de sobreposição temporal dos registos no [DNSSec](#), este último associado à respectiva substituição dos certificados. Seguidamente, apresentam-se as expressões do *script* *cacert* que implementam a decisão dos estados com base na existência dos certificados da [CA](#). As expressões são retiradas do ciclo principal do *script*. No mesmo directório, a partir da primeira vez que o *script* é executado, existe sempre um certificado da [CA](#) actual, para o momento. Durante o período de substituição de registos, existe para além disso um certificado da [CA](#) para o momento seguinte, com uma nova chave. No fim do período, este último certificado passa a ser o certificado actual do nó.

```
if [ ! -f ${TOPDIR}/ScalSecCA/ScalSecCrt.pem -a ! -f ${TOPDIR}/ScalSecC
```

```

A/ScalSecCrtNext.pem ]; then
# Momento inicial ou após remover a CA criada antes
    else
        if [ ! -f ${TOPDIR}/ScalSecCA/ScalSecCrtNext.pem ]; then
# Momento para finalizar a substituição do registo e
# substituição dos certificados nos serviços
            else
#Momento para iniciar a substituição do registo
                fi
            fi

```

4.3.3 Actualização dinâmica no DNS

O *script* *cacert* permite realizar a actualização dinâmica dos registos H no [DNSSec](#). A actualização é realizada pela ferramenta *nsupdate*, permitindo adicionar os registos novos e remover os antigos. Para utilizar o *nsupdate* é necessário que o [DNSSec](#) esteja configurado para permitir alterar os registos associados ao [FQDN](#) do nó, e que seja criada uma chave [TSIG](#) para realizar as alterações de forma segura. As políticas de actualização dinâmica do [DNSSec](#), associadas ao nome do nó, têm de ser configuradas por quem é responsável pelos servidores de nomes de domínio. Terá que ser disponibilizada, normalmente, uma chave ao administrador do nó por cada [FQDN](#).

De seguida apresentam-se os comandos utilizados no *script* para adicionar um registo do tipo H.

```

echo "prereq yxrrset $HOSTNAME IN A" > $TOPDIR/ScalSecCA/$NSUPDATEF
echo "update add 'cat < ${TOPDIR}/ScalSecCA/$TYPE120RRF'" >> $TOPDIR/ScalSecCA/$NSUPDATEF
echo "send" >> $TOPDIR/ScalSecCA/$NSUPDATEF
nsupdate -d -k $KEYFILE -v $TOPDIR/ScalSecCA/$NSUPDATEF

```

Para remover um registo H, o modo é idêntico, com o comando adicional para testar que de facto o registo a remover se encontra efectivamente no [DNSSec](#). Em qualquer um dos casos, é realizado o teste de que de facto se encontrava um registo A com o nome do nó.

4.3.4 Gestão e agendamento temporal

O uso de um *script* para gerar os certificados e realizar a actualização dinâmica do [DNSSec](#), por si só, é uma mais valia. Permite automatizar o uso dos comandos e manipulação dos ficheiros, permitindo uma relativa abstracção para o administrador do nó. Contudo para uma maior optimização, é necessário recorrer a uma gestão temporal, para permitir agendar os momentos para os quais o *script* deverá correr, de forma automática.

Para agendar processos no *Ubuntu* existem várias opções, cada uma delas com as suas particularidades, na forma como colocam os processos em agenda, e na forma como

o processo é chamado no momento agendado. A utilização do *crontab* de utilizador é, possivelmente, a forma mais conhecida para o fazer. Outra é a utilização do comando *at*. O *crontab* é bastante flexível e permite agendar as tarefas não só para um determinado momento, mas também consecutivamente de forma cíclica, para vários momentos. O comando *at* permite apenas agendar para um determinado momento absoluto.

Para o caso da implementação o comando *at* revelou-se menos problemático, e mais indicado, pela sua simplicidade e robustez na forma como o processo é agendado. Os momentos em que o *script* deve correr são determinados pelo próprio *script*, baseados nas variáveis temporais em jogo. O *script* agenda apenas para o próximo momento, no futuro, no qual o mesmo deve ser chamado novamente. Entretanto, até esse momento, o mesmo poderá ser chamado manualmente, não inviabilizando a automatização, e actualizando o novo momento para o qual é agendado.

De seguida apresentam-se as expressões para agendar os momentos para os quais os *scripts* deviam correr.

```
for job in $(atq | cut -f1); do
    if at -c $job | tail | grep $TOPDIR/cacert ;then
        atrm $job
    fi
done
DATE='date -d "$DAYS days -$TTL seconds" "+%H:%M %m%d%y"'
echo "$TOPDIR/cacert" | at $DATE
```

Inicialmente, realiza-se o teste para verificar se já existe uma entrada agendada. Caso isso seja verdade, esta é removida. Apenas pode existir uma entrada, em qualquer momento, referente ao nome do *script*, seja ele o principal ou o secundário. Seguidamente, é adicionada a nova entrada com o novo momento. O tempo é calculado relativamente ao tempo local do nó, e de acordo com o estado actual.

Outro aspecto significativo é a capacidade de actualização automática dos certificados nos respectivos serviços a disponibilizar. Normalmente, é necessário ter em conta a actualização do certificado da [CA](#) do nó, do certificado e da respectiva chave do serviço. As operações envolvidas são a simples substituição ou concatenação de ficheiros. É necessário reiniciar os serviços logo após a substituição dos certificados. Para a actualização automática dos certificados nos respectivos serviços usa-se um ficheiro, *servicedata*, com a indicação dos serviços, os ficheiros a actualizar, e o comando de reinicialização do serviço. Para cada serviço, é usada uma linha no *servicedata* e os campos são delimitados por caracteres de tabulação. Seguidamente, apresenta-se o exemplo, da linha, para o caso do serviço no porto 443:

```
__tcp_443    /etc/apache2/ssl/ScalSecCrt.pem    /etc/apache2/ssl/__tcp_443_c
rt.pem      /etc/apache2/ssl/__tcp_443_key.pem  /etc/init.d/apache2 force-re
load
```

4.4 Conclusões

Os certificados melhorados foram gerados manualmente. Posteriormente optimizou-se o processo para gerar os certificados mais rapidamente, com a adição automática ao [DNSSec](#) do registo H. Foi implementado um processo que permite a respectiva gestão temporal da substituição de certificados e registos no [DNSSec](#).

O custo de obtenção de certificados segundo o novo modelo é semelhante à emissão de certificados *self-signed*, permite maior controlo, por parte de quem gere o nó, na certificação dos respectivos serviços, em relação ao modelo [PKIX](#).

A solução permite um controlo maior sobre os certificados ao nível dos servidores. Não existe modificação na implementação dos serviços ou aplicações. A alteração é realizada apenas nos certificados usados. A facilidade e rapidez com que se pode gerar os certificados é uma grande vantagem em relação ao [PKIX](#). Com a implementação através de scripts ou até de um processo é possível ter um bom nível de abstracção da gestão dos certificados.

Capítulo 5

Cliente

Neste capítulo descreve-se a implementação e configuração do cliente [SSL/TLS](#) (dora-vante apenas referido como cliente [TLS](#)). O cliente [TLS](#) foi alterado de forma a suportar o [PKIX](#) e também o [ScalSec](#). A alteração implicou o recurso a um *resolver* [DNSSec](#), implementado pelo [LWRES](#) (*Lightweight Resolver Library*), e a alterações das bibliotecas do *Mozilla*, principalmente ao nível do [NSS](#). O cliente [TLS](#) é comum a duas aplicações, o *Firefox* e o *Thunderbird*, clientes de *Web* e de email, os quais são serviços importantes.

5.1 Cliente TLS do Projecto Mozilla

As principais alterações para a implementação da solução estão localizadas no módulo [NSS](#). Este módulo está incluído nos projectos de segurança (*Security Projects*), localizados no código fonte, no directório `mozilla/security`. Este código é incluído nas várias aplicações e está profundamente associado a outros módulos.

Na figura 5.1 pode-se observar as bibliotecas relevantes para a implementação do cliente [TLS](#). O [PSM](#) (*Personal Security Manager*) consiste num conjunto de bibliotecas que realizam operações de criptografia por parte de uma aplicação de cliente. Uma das operações é o estabelecimento de sessões [TLS](#), para as quais depende do [NSS](#). O [NSS](#) abrange um conjunto de bibliotecas que suportam o desenvolvimento de aplicações seguras, tanto do lado do cliente como do servidor. É ao nível do [NSS](#) que são implementadas as funções que realizam o *handshake* do [TLS](#) e as operações de verificação dos certificados, típicas do modelo [PKIX](#). Para além disso, existe o [NSPR](#) (*Netscape Portable Runtime*), que não está relacionado com aspectos de segurança mas fornece uma plataforma de integração para vários ambientes de sistemas operativos. O [NSPR](#) facilita serviços de gestão ao nível de ficheiros, memória, rede e bibliotecas partilhadas, entre outros. É o [NSPR](#) que faz a interface com as bibliotecas de resolução de nomes, externas ao código *Mozilla*. Contudo, não está incluída a capacidade de resolução segura recorrendo ao [DNSSec](#). O [LWRES](#) tem essa capacidade, e pode ser adicionado ao nível do [NSPR](#), para ser utilizado no [NSS](#). Actualmente, o [LWRES](#) só é utilizado para efeitos de [TLS](#). A resolução inicial dos nomes

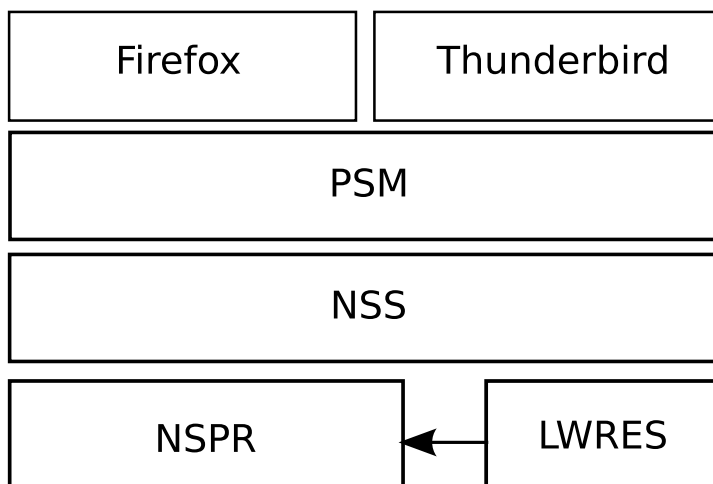


Figura 5.1: Relação das bibliotecas para a implementação do cliente TLS.

em endereços IP continua a ser feita através do resolver [DNS](#) convencional. Contudo, a actual integração do [LWRES](#) permite uma futura resolução dos referidos recorrendo ao [DNSSec](#).

5.2 Verificação dos certificados X.509

Para suportar o modelo de certificação ScalSec, o cliente teve de ser alterado de forma a processar a verificação dos certificados de forma diferente do [PKIX](#):

1. Verificar se o identificador que vem no certificado do serviço, do tipo `__proto_porto` coincide com o protocolo de transporte utilizado e o porto onde a aplicação servidor (serviço) escuta no nó servidor.
2. Verificar se o certificado do serviço foi emitido pela [CA](#), que é o nó servidor, as assinaturas e períodos de validade estão correctos.
3. Verificar se existe um mapeamento seguro entre o [FQDN](#) ou endereço [IP](#), que o utilizador introduziu, e a chave pública contida no certificado ([CA](#)) que representa o nó servidor.

De acordo com o modelo de certificação ScalSec, para que isto seja possível recorre-se ao [DNSSec](#).

Até à recepção dos certificados no cliente não existe qualquer modificação no código. As primeiras alterações foram realizadas na função `SSL_AuthCertificate()`:

`SECStatus`

```
SSL_AuthCertificate(void *arg, PRFileDesc *fd, PRBool checkSig,
                  PRBool isServer)
```

A função encontra-se na biblioteca [SSL](#) do [NSS](#) no ficheiro `sslauth.c`; é chamada quando as mensagens com os certificados são recebidas. Onde era chamada a função que desencadeia o processo de validação dos certificados e validade da cadeia dos mesmos

Do certificado do servidor extrai-se o [DN](#) do emissor e verifica-se se a organização é `ScalSec`. Se for o caso, de seguida obtém-se a informação acerca do porto da conexão ao servidor.

Ainda na função `SSL_AuthCertificate()` é invocada a função `CERT_VerifyCertName()`, que compara o nome do servidor com o nome presente no certificado. Isto acontece depois da verificação da validade do certificado. O código foi alterado de forma a que essa comparação apenas é realizada na implementação original.

Outro ficheiro alterado, `certvfy.c` contém grande parte das funções utilizadas para a verificação dos certificados. Seguindo a lógica da estrutura das bibliotecas do [NSS](#) a função `CERT_VerifyScalSecNow()` foi criada no mesmo ficheiro de modo semelhante à função `CERT_VerifyCertNow` já existente, a qual recebe nos argumentos o nome do servidor e do porto da conexão, o qual pode variar, mas por omissão é o 443 para o [HTTP](#), o 995 para o [POP3S](#) ([Post Office Protocol Secure](#)), o 993 para o [IMAPS](#) ([Internet Message Access Protocol Secure](#)), e o 25 para o [SMTP](#) (com `STARTTLS`).

`SECStatus`

```
CERT_VerifyScalSecNow(CERTCertDBHandle *handle, CERTCertificate *cert,
                      PRBool checkSig, SECCertUsage certUsage,
                      void *wincx, char* url, unsigned short port)
{
    return(CERT_VerifyScalSec(handle, cert, checkSig, certUsage,
                              PR_Now(), wincx, NULL, url, port));
}
```

A função `CERT_VerifyScalSec()` é semelhante à função `CERT_VerifyCert()`. A principal diferença é a invocação da `CERT_ScalSec()`. Nos argumentos recebidos, inclui-se o nome do servidor e o porto da conexão para posterior verificação.

`SECStatus`

```
CERT_VerifyScalSec(CERTCertDBHandle *handle, CERTCertificate *cert,
                  PRBool checkSig, SECCertUsage certUsage, int64 t,
                  void *wincx, CERTVerifyLog *log, char* url,
                  unsigned short port)
```

Esta função realiza a validação do certificado do servidor (aplicação) e invoca a função para validação da cadeia. Tal como a função correspondente para o [PKIX](#), realiza as operações básicas de validação, tais como verificar o período de validade dos certificados.

A comparação do porto presente no certificado do servidor com o da conexão faz-se de acordo com a seguinte descrição: se a igualdade não for verificada, o certificado é

declarado inválido e o utilizador é notificado através de uma mensagem de erro, indicando que o certificado não é de confiança. No excerto que se segue é possível confirmar a referida validação e eventual notificação de erro.

```
SCommonName=CERT_GetCommonName(&cert->subject);
prtclcert=strtok_r(SCommonName,"_", &saveptr);
portcert=strtok_r(NULL, "_", &saveptr);
portcertn= (unsigned short) PORT_Atoi(portcert);
if(port!=portcertn){
    PORT_SetError(SEC_ERROR_UNTRUSTED_CERT);
    rv=SECFailure;
    return rv;
}
PORT_Free(SCommonName);
```

Após a validação do certificado do serviço, inicia-se a validação da cadeia de certificados. A função `CERT_ScalSec()` faz a referida validação, e é semelhante à função `CERT_VerifyCertChain()`; inclui a passagem do argumento do nome do servidor já referido.

```
SECStatus
CERT_ScalSec(CERTCertDBHandle *handle, CERTCertificate *cert,
             PRBool checkSig, SECCertificateUsage certUsage,
             int64 t, void *wincx, CERTVerifyLog *log, char* url)
{
    return cert_ScalSec(handle, cert, checkSig, NULL, certUsage,
                        t, wincx, log, NULL, url);
}
```

Para além das funções disponíveis no código *Mozilla* apenas é necessário usar a função `getrrsetbyname` do [LWRES](#). A comparação dos resumos das chaves públicas é realizada em formato binário. Comparam-se os tipos de algoritmo e resumo. A validação dos registos presentes no [DNSSec](#) depende da configuração do [LWRES](#) (*Lightweight Resolver Library Daemon*), pode ser realizada na máquina do cliente ou num servidor local. Neste último caso a segurança na transmissão dos registos e da informação de validação é feita através do [TSIG](#).

A solução suporta a utilização de endereços [IP](#) como identificador introduzido pelo utilizador: o pedido ao registo H é realizado na zona reversa do [DNS](#). A solução suporta IPv4 e IPv6. Para verificar se o fornecido pelo utilizador é um endereço [IP](#) recorre-se à função `PR_StringToNetAddr()`:

```
if(PR_StringToNetAddr(url, &netAddr)==PR_SUCCESS){
```

Posteriormente, é invocada a função para obtenção segura do registo tipo H (`rdtype=0x78`).

```
result=getrrsetbyname(host, rdclass, rdtype, flags, &res);
```

Se a validação dos registos falhar, é apresentada uma mensagem de erro de [CA](#) não confiável ao utilizador:

```
if(!(res->rri_flags & RRSET_VALIDATED)){
    PORT_SetError(SEC_ERROR_UNTRUSTED_ISSUER);
    LOG_ERROR_OR_EXIT(log, issuerCert, 1, 0);
}
```

No caso de ser obtido uma ou mais registos tipo H associados ao nome/endereço associado ao servidor, obtém-se o tipo de chave presente no certificado. Para cada registo, verifica-se se o tipo de chave da qual foi obtida o resumo é o mesmo que o da chave no certificado do nó. Depois, calcula-se o resumo da chave presente no certificado de acordo com o tipo de resumo declarado no registo. Por fim, os resumos são comparados e, se a igualdade se verificar, declara-se como correcta a validação da cadeia de certificados. Se não for encontrado nenhum registo que verifique a igualdade ou por qualquer outro motivo se chegar ao final da função sem validação objectiva da cadeia dos certificados, é declarado o erro de emissor desconhecido.

Se a validação não for positiva, o utilizador é notificado com uma mensagem de erro de [CA](#) não confiável:

```
PORT_SetError(SEC_ERROR_UNKNOWN_ISSUER);
LOG_ERROR(log, subjectCert, 0, 0);
```

5.3 Validação do mapeamento do nome/endereço no nó servidor

O módulo [NSS](#) não inclui um resolver [DNSSEC](#), logo é necessário implementar ou recorrer a um já existente. Numa primeira fase recorreu-se ao resolver [DNSSEC DIG](#) através de chamadas ao sistema.

```
sprintf(DigCmd, "dig %s +dnssec +sigchase +multiline -t DNSKEY
> digfile", hostname);
system(DigCmd);
```

A validação do nó fez-se com a implementação de um ciclo que realizava os pedidos de informação aos servidores de [DNSSEC](#) utilizando o [DIG](#). Recorreu-se ao seguimento das assinaturas (sigchase) ate encontrar uma chave de confiança. O ciclo considerava o suporte de sinónimos de nomes ([CNAME](#)). O pedido inicial fazia-se por qualquer registo associado ao nome assumido pelo servidor, um registo [CNAME](#) ou A. Após encontrar um registo A fazia-se o pedido para a obtenção da informação de chave pública associada ao nó. A informação de cada pedido era registada num ficheiro temporário para análise. Se o nome

do servidor não fosse obtido a validação do nó falhava. Depois de cada pedido fazia-se a reinicialização das *flags* e realizava-se o processamento da informação obtida. Após a abertura do ficheiro temporário realizava-se a obtenção da informação necessária.

A obtenção da chave pública do nó era realizada quando esta era detectada de acordo com a formatação dos dados presentes no ficheiro. Por cada ficheiro de um pedido ao [DIG](#) fazia-se a recolha de informação do registo em causa, para decisão em relação ao próximo registo a ser pedido. Depois de toda a verificação da validade dos dados obtidos, restava comparar as chaves públicas. As chaves teriam que ser iguais para que a validação fosse sinalizada como correcta à biblioteca [SSL](#) do módulo [NSS](#). Posteriormente, no caso presente a validação seria sinalizada para o [PSM](#).

O processo de validação implementado e descrito anteriormente foi então optimizado. As chaves públicas envolvidas foram substituídas pelos respectivos resumos, permitindo a diminuição do tamanho da informação transmitida pelos servidores [DNS](#). Por outro lado implica o cálculo adicional do resumo da chave no cliente. Mais tarde foi feita uma melhor integração recorrendo à biblioteca [LWRES](#).

5.3.1 LWRES

A opção de utilizar a biblioteca [LWRES](#) justifica-se por varias razões. Trata-se de uma biblioteca aliada à implementação do [BIND](#). A biblioteca [LWRES](#) possui um conjunto de funções, as quais recorrem ao *daemon* [LWRES](#)[D](#). Algumas funções disponibilizadas permitem a substituição das funções que tipicamente desencadeiam a normal resolução de nomes (`netdb.h`). No ficheiro `netdb.h` do [LWRES](#) (`lwres/netdb.h`) estão definidas todas as funções do anterior. Não é necessário alterar o nome das funções já existentes. O [LWRES](#) especifica novas funções, de entre as quais uma delas é a usada na implementação (`getrrsetbyname`).

O [LWRES](#)[D](#) é um servidor de nomes de domínio que recebe pedidos através do protocolo [LWRES](#) em vez do protocolo [DNS](#). Alternativamente ao uso do [LWRES](#)[D](#), pode-se configurar o [BIND](#) para funcionar como o [LWRES](#)[D](#). As opções usadas para configuração são semelhantes. Por omissão as aplicações que usam [LWRES](#) comunicam através de pedidos por [UDP](#) (*User Datagram Protocol*) ao [LWRES](#)[D](#) no endereço de *loopback* no porto 921. As aplicações usam portos não privilegiados para os pedidos.

5.4 Interface com o utilizador

Os erros a passar ao [PSM](#) são praticamente os mesmos. Não foram adicionadas novas mensagens de erro, usam-se as já existentes, declaradas no código original. Considerar erros específicos só teria significado no caso de apresentar informação específica ao utilizador, o que implicava a reestruturação profunda da interface gráfica. A apresentação de informação específica do [DNSSec](#) ao utilizador é discutível. A nossa opinião é que o utilizador comum não entenderia o significado dessa informação, dando margem a potenciais



Figura 5.2: Exemplo de erro de validação com ScalSec no Firefox.

ataques. Nas figuras 5.2 e 5.3 observam-se os casos em que ocorre, respectivamente, um erro e o sucesso de validação, no *Firefox*, para a solução baseada no ScalSec. Na figura 5.4 observa-se a configuração do POP3 no *Thunderbird* para o ScalSec. Como se pode observar, para a solução baseada no ScalSec, a interface gráfica e a configuração no cliente, não se altera em relação ao PKIX.

5.5 Compilação e instalação dos Clientes

A descrição e instalação dos clientes é comum, quer para o *Firefox*, quer para o *Thunderbird*. Os repositórios no *Ubuntu* disponibilizam o código fonte original e um ficheiro *diff*, por questões de compatibilidade. Para produzir a aplicação é necessário aplicar o *diff* ao código fonte original do directório obtido a partir do arquivo. Tal pode ser realizado da seguinte forma, automaticamente, para o caso do *Firefox*:

```
$ apt-get source firefox
```

Para além do arquivo com o código obtém-se uma directório (*debian*) com ficheiros para a compilação e instalação da aplicação. Os *patches* que fazem parte desse directório devem ser aplicados da seguinte forma:

```
$ tar -xjf firefox-2.0.0.14+nobinonly-source.tar.bz2
```

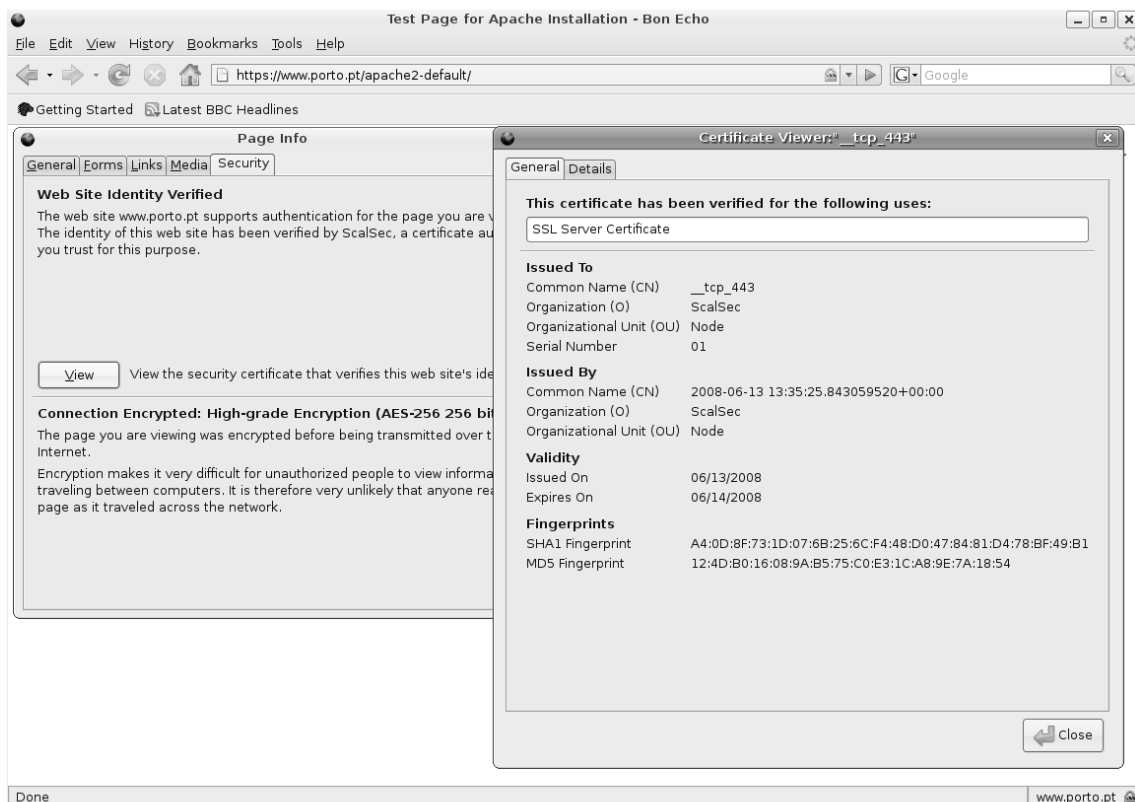


Figura 5.3: Exemplo de sucesso na validação com ScalSec no Firefox.

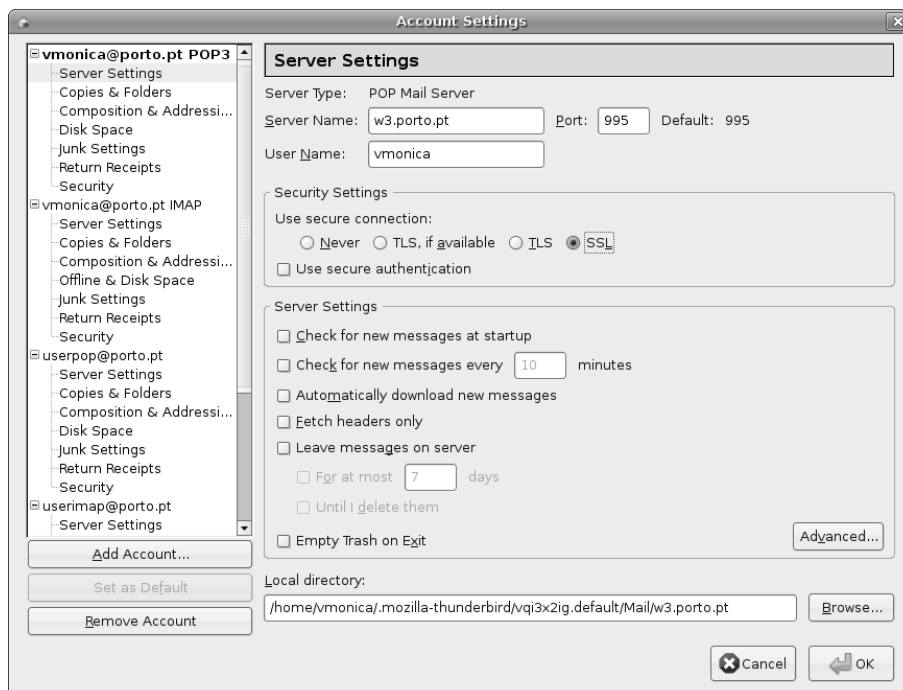


Figura 5.4: Exemplo de configuração para ligação POP3 com ScalSec, no Thunderbird.

```
$ export QUILT_PATCHES=/home/vmonica/Projects/
firefox-2.0.0.14+2nobinonly/debian/patches
$ cd mozilla/
$ quilt --quiltrc /dev/null push -a || test $? = 2
```

O código fonte original permite obter diferentes aplicações em diversas plataformas de sistemas operativos. Para a produção de uma aplicação recorre-se às ferramentas *autoconf* e *make*, utilizadas tradicionalmente em sistemas *Unix*. Contudo, estas ferramentas são utilizadas mesmo em sistemas não baseados em *Unix*. Não é recomendável correr os comandos *configure* e *make* com as opções por omissão porque não produz uma aplicação óptima, funcionalmente, para a plataforma em causa. É possível fazer a chamada de *configure* com as opções de configuração na linha de comandos, mas é aconselhável recorrer a um ficheiro de configuração. O caminho do ficheiro com as opções desejadas deve ser exportado para a variável de ambiente *MOZCONFIG*. As opções de configuração podem ser obtidas a partir de uma versão já instalada na plataforma colocando a seguinte linha no campo de endereços do *Firefox*, ou na opção de definição de página inicial para o *Thunderbird*:

```
about:buildconfig
```

Após reunir as opções necessárias no ficheiro de configuração, exporta-se o caminho do próprio e pode-se compilar o código. O executável encontra-se no directório *dist/bin*.

```
$ export MOZCONFIG=/home/vmonica/Projects/mozconfigfiles/
firefox-2.0.0.14-mozconfig
$ make -f client.mk build
```

O código compilado é organizado por directórios, de modo semelhante ao código fonte, e possui *makefiles*. É possível alterar o código posteriormente e compilar apenas os módulos necessários. Para o caso da implementação era necessário compilar adicionalmente os módulos alterados com a biblioteca *LWRES*. A biblioteca foi previamente instalada. Não foi integrada no código devido a complexidade da estrutura do mesmo com as *makefiles* distribuídas. A opção mais simples foi exportar a referência à biblioteca para uma das variáveis que especificava as bibliotecas para o código:

```
$ export EXTRA_LIBS="-llwres"
$ make
```

5.6 Instalação do LWRES D

Para a implementação foi utilizado o *LWRES D* configurado de acordo com o que se pretendia, recorrendo ao ficheiro de configuração */etc/bind/lwres.conf*. Para activar o suporte do *DNSSec* e a obtenção de informação de validação, as seguintes opções devem estar presentes no ficheiro de configuração:

```
dnssec-enable yes;
dnssec-validation yes;
```

Normalmente as opções de *forwarding* para o caso de uma máquina local que faz uso de um servidor recursivo devem ser configuradas para o mesmo. Deve ser indicado o endereço do servidor de [DNS](#) recursivo. Se a máquina local não possuir a resposta em cache, então faz o pedido de resolução ao servidor recursivo, sem recurso a pedidos a servidores externos. Tal configuração tem vantagens para a criação de cache ampla ao nível de uma rede interna. Reduz-se o tráfego para o exterior da mesma rede interna.

```
forward only;
forwarders {
    172.16.10.7;
};
```

Para activar o uso do protocolo [LWRES](#) basta declarar a sua utilização. Não é necessário configurar com qualquer argumento.

```
lwres{};
```

Deve existir uma chave para que a cadeia de validação do [DNSSEC](#) seja verificada.

```
trusted-keys {
    "pt." 256 3 5 "AQ0v3wVb2k6LsSz/56jL60kCQ2eke6YcFksI7cagUiBA/AwuJyb8X0Ii
        g09EcZ1SSAu1NB6oQLNidZviX5+1D8G4LWkLH+bwYrhmKILn5qc6WuTA
        tIKHwgtAH+x3e+Tt8ps=";
};
```

Para permitir a transacção segura dos registos do servidor recursivo para o cliente pode-se usar [TSIG](#). A transacção segura dos registos é necessária uma vez que os registos podem ser alterados pelo caminho, bem como a indicação de validação [DNSSEC](#), a qual é baseada numa *flag*, o bit [AD](#) (*Authenticated Data*). O [TSIG](#) é baseado em [MAC](#) (*Message Authentication Codes*), e permite autenticar os pedidos e respectivas respostas com validade temporal relativa. Para tal deve ser criada uma chave simétrica. Esta chave deve ser colocada na configuração quer do cliente quer do servidor [DNS](#).

```
key valter.porto.pt.{
    algorithm hmac-md5;
    secret "Me9gkynH0t1JViC6EfFpNA==";
};
```

No cliente deve-se indicar que para o servidor [DNS](#) (neste caso com endereço 172.16.10.7) a respectiva chave é usada para assinar os pedidos. No servidor [DNS](#) as respostas são assinadas com a respectiva chave se os pedidos também o forem.

```
server 172.16.10.7{  
    keys{valter.porto.pt. };  
};
```

Este mecanismo de obtenção segura dos registos foi posteriormente testado recorrendo a um pequeno programa em C, o qual permitiu analisar e verificar o funcionamento do [LWRESD](#) de acordo com as opções presentes na configuração. Permitiu também analisar a forma como a informação dos registos e dados adicionais é obtida.

5.7 Conclusão

A implementação no cliente permitiu obter os registos do [DNSSec](#) de forma segura. O cliente [TLS](#) foi modificado para implementar a nova solução em paralelo com o modelo [PKIX](#).

Capítulo 6

Análise de resultados

Depois da implementação da solução para o modelo considerado, foi realizada uma análise da solução. A análise de desempenho da solução em comparação com a actual solução para o [PKIX](#) surge naturalmente. Para além da análise de desempenho, faz-se a análise dos certificados envolvidos. Os certificados X.509 são utilizados nos dois modelos de autenticação. Os requisitos para a informação presente nos mesmos são no entanto diferentes. A análise foi realizada em grande parte com recurso ao *Wireshark*. Foram observadas as mensagens entre cliente e servidores, do lado do cliente. Foi dada particular atenção aos tempos relativos de processamento da informação recebida e ao tamanho da mesma. A análise é composta por duas partes, a primeira é sobre o desempenho do cliente, a segunda é sobre o tamanho dos certificados enviados aquando do estabelecimento das sessões [TLS](#), durante o *handshake*.

6.1 Desempenho

Para a análise de desempenho foi dada particular atenção aos tempos de processamento envolvidos no lado do cliente. O objectivo foi a comparação da solução implementada com a implementação do [PKIX](#). No caso da solução implementada foi utilizada uma testbed com servidores [DNSSec](#), e um servidor Web dedicados. O servidor é o `www.porto.pt`, que é um nome sinónimo ([CNAME](#)) de `w3.porto.pt`. Na figura [6.1](#) podem ser observadas as mensagens relevantes para a análise do novo modelo.

Para o [PKIX](#) utilizámos a *Internet*, o servidor *Web* é o `www.fe.up.pt`. Neste caso o cliente recorre a um *proxy HTTP* para a ligação ao servidor *Web*. Contudo as mensagens relevantes para análise surgem de igual forma. Na figura [6.2](#) podem-se observar as mensagens relevantes na análise do modelo [PKIX](#).

Seria desejável realizar uma análise de desempenho da solução considerando cenários mais próximos dos reais. Contudo, tal análise envolve várias variáveis difíceis de controlar e observar. Uma das variáveis é a cache de informação nas diversas máquinas. A influência da cache é difícil de quantificar. Devido a este facto, não se considerou a cache [DNS](#) nem

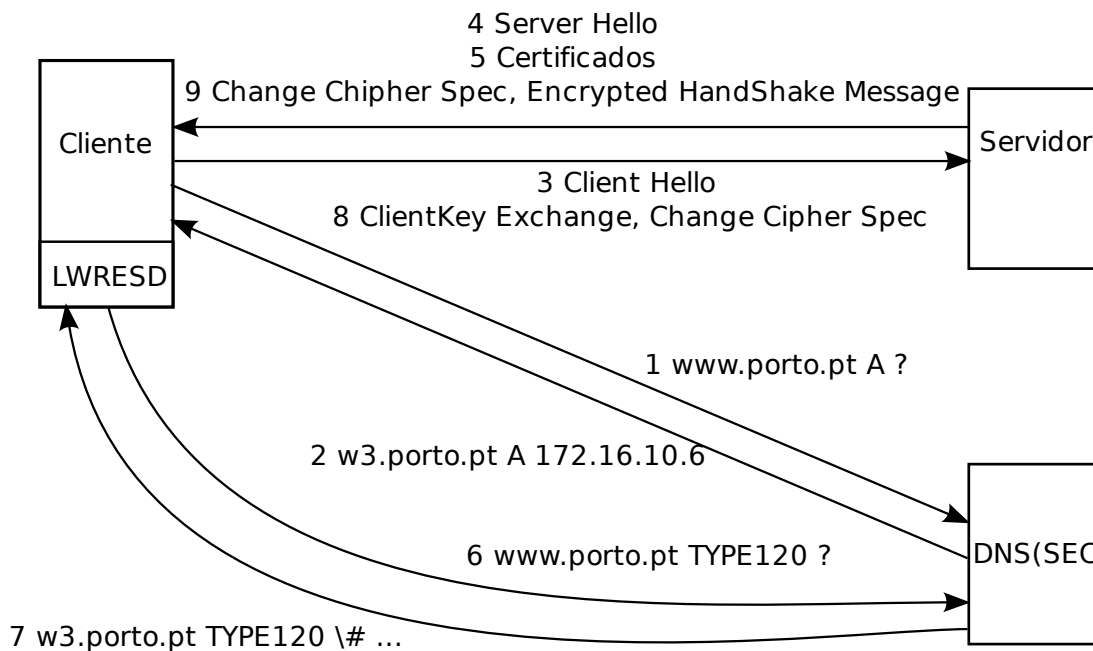


Figura 6.1: Mensagens relevantes na autenticação na nova solução.

[HTTP](#) nas máquinas, a máquina cliente foi sujeita a limpeza de cache da informação entre medidas.

Numa primeira fase foram obtidas medidas de tempo relativo entre um pedido [DNS](#) para resolução do nome do servidor em endereço (registo A) e respectiva resposta. Os tempos são relativamente idênticos. Os valores medidos podem ser observados na tabela [6.1](#). Não favorecem nem prejudicam nenhuma solução analisada.

Para os dois modelos, o estabelecimento de uma sessão [TLS](#) faz-se de igual modo até ao momento em que o cliente recebe os certificados do servidor *Web*. Após esse momento, inicia-se o processo de autenticação do servidor *Web*, quer seja no [PKIX](#), quer seja na solução implementada. No caso do [PKIX](#), o processamento no cliente estende-se até um pedido ao [OCSP](#). No caso do novo modelo estende-se até um pedido através do [LWRES](#) para obtenção do registo H (TYPE120). Estes tempos foram registados e podem ser observados na tabela [6.2](#). Observa-se que o tempo médio no novo modelo é menor, mas são processos diferentes de validação. Os dois modelos processam-se de forma diferente. Para este menor atraso contribui o facto do ScalSec se basear em certificados, de menor tempo de vida e consequentemente chaves e assinaturas menores, logo um menor tempo de transferência. Contudo, este tempo não deve ser considerado representativo de todo o processo de validação.

Outro tempo de processamento no cliente a considerar é o tempo entre a resposta aos pedidos referidos anteriormente, registo H ou [OCSP](#), e o final do processo de autenticação. A tabela [6.3](#) apresenta os valores observados. Em média, o tempo para o novo modelo,

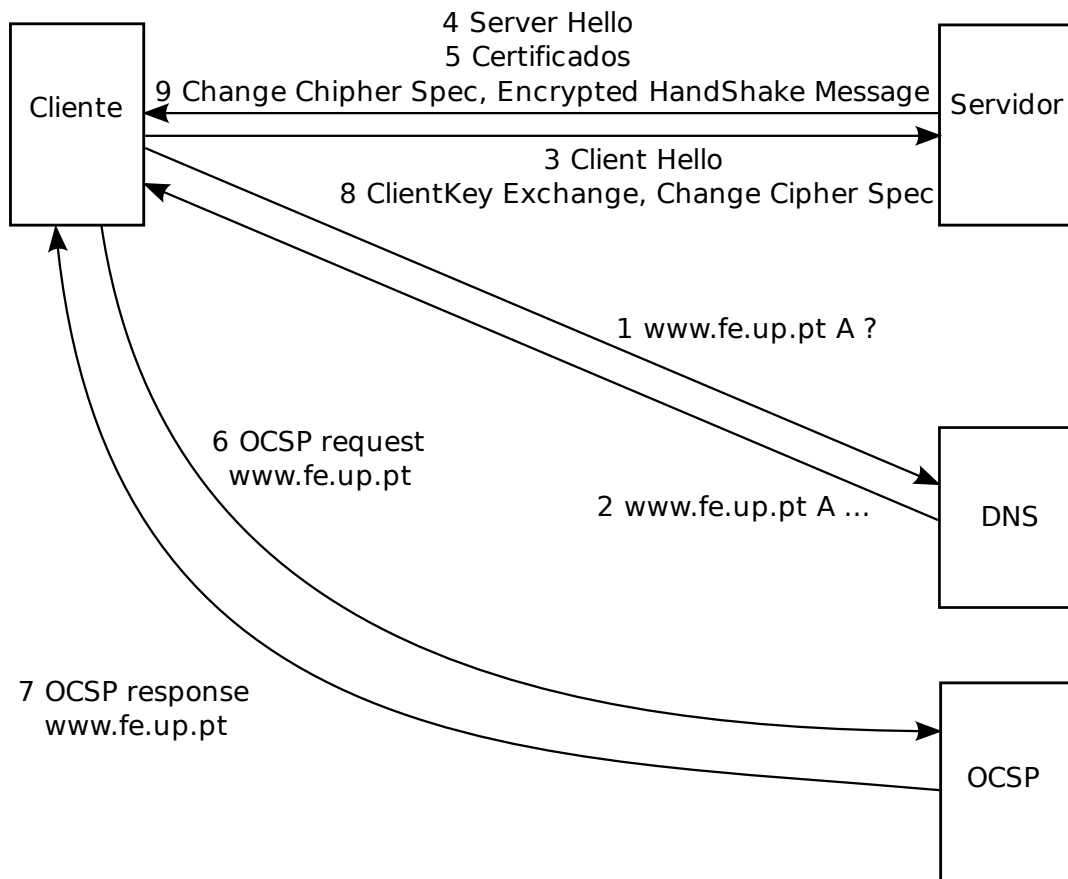


Figura 6.2: Mensagens relevantes na autenticação no modelo PKIX.

Tabela 6.1: Tempos entre pedido e resposta do DNS de um registo A.

	Sítio	
	www.porto.pt	www.fe.up.pt
Medidas (segundos)	0.000631	0.000608
	0.000603	0.000603
	0.000576	0.000576
	0.000618	0.000620
	0.000640	0.000600
	0.000602	0.000602
	0.000595	0.000595
	0.000633	0.000633
	0.000623	0.000623
	0.000609	0.000609
Médias	0.000613	0.000607

Tabela 6.2: Tempos de processamento do cliente, entre obtenção dos certificados e os pedidos LWRES ou OCSP, consoante se trata de ScalSec ou PKIX, respectivamente.

	Tempo de processamento (segundos) entre recepção de	
	certificados e pedido LWRES	certificados e pedido OCSP
Medidas	0.040693	0.120412
	0.038625	0.121082
	0.051040	0.116732
	0.051875	0.115539
	0.043229	0.122803
	0.035081	0.121969
	0.046239	0.121896
	0.087131	0.118597
	0.034465	0.117986
	0.035334	0.117611
Médias	0.046371	0.119463

Tabela 6.3: Tempos de processamento entre resposta aos pedidos LWRES ou OCSP e final da validação por parte do cliente.

	Tempo de processamento (segundos) entre	
	LWRES e Client Key Exchange	OCSP e Client Key Exchange
Medidas	0.025592	0.043351
	0.022941	0.015663
	0.101979	0.015466
	0.021604	0.043241
	0.021518	0.015587
	0.089389	0.015588
	0.024077	0.034138
	0.021335	0.035048
	0.020636	0.043632
	0.020576	0.017260
Médias	0.036965	0.027897

Tabela 6.4: Total dos tempos de processamento na validação do lado do cliente.

Total do tempo de processamento (segundos)		
	ScalSec	PKIX
Medidas	0.066285	0.163763
	0.061566	0.136745
	0.153019	0.132198
	0.073479	0.158780
	0.064747	0.138390
	0.124470	0.137557
	0.070316	0.156034
	0.108466	0.153645
	0.055101	0.161618
	0.055910	0.134871
Médias	0.083336	0.147360

é relativamente superior ao tempo para o modelo [PKIX](#). Isto explica-se pelo facto de no novo modelo, nesta fase de processamento, se realizar o cálculo de resumos de chaves, de acordo com a informação que é recebida através do [LWRES](#).

Para o novo modelo é necessário processar a informação recebida para proceder à validação dos certificados. Este tempo pode até ser superior se estiverem presentes mais do que um registo H no [DNSSec](#). Para a análise foi considerado que apenas um estava presente. Em caso de troca de chaves no servidor, podem estar presentes dois registos H no [DNS](#), mas durante um tempo reduzido. A informação está distribuída e pode ficar em cache o que permite diminuir o tempo para obter a informação pelo [LWRES](#).

Para o modelo [PKIX](#) resta processar a informação recebida do [OCSP](#) para verificação de revogação dos certificados. O [OCSP](#) permite confirmar a revogação dos certificados com mensagens curtas encapsuladas em [HTTP](#). Contudo, tipicamente, o cabeçalho [HTTP](#) apresenta tamanhos consideráveis em comparação com o [OCSP](#). Torna-se necessário fazer a resolução de nome para o servidor [OCSP](#).

Para a contabilização dos tempos de processamento no cliente foi realizada a soma dos tempos dos dois períodos referidos, nomeadamente . Esses valores podem ser observados na tabela 6.4. De um modo geral, no novo modelo, o tempo de processamento no cliente é menor. O processamento adicional e tempo que é necessário para obtenção segura do registo H não vai degradar a autenticação da sessão [TLS](#). Este processamento e tempo depende de alguns factores e pode variar se existe ou não mais do que um registo H. A configuração do serviço [DNSSec](#) contribui para esses tempos, não se referindo à cache.

6.2 Tamanho dos certificados transferidos

O tráfego [DNSSec](#) para resolução do registo H, bem como o tráfego para resolução do *OCSP Responder* e consequentemente tráfego [OCSP](#) podem contribuir para a avaliação

Tabela 6.5: Número de certificados enviados para o cliente TLS no modelo PKIX.

Sítios	Número de certificados recebidos
www.fe.up.pt	2
www.bes.pt	3
www.paypal.com	3
www.google.com	2
caixadirecta.cgd.pt	2
launchpad.net	2
www.alertpay.com	2
signin.ebay.com	3
www.mbnet.pt	3
www.amazon.com	2

em termos de tráfego. Pelo facto de haver muitas variáveis, entre elas a probabilidade, dos referidos registos [DNS](#) e resposta [OCSP](#) estarem em cache, não foi possível fazer uma avaliação rigorosa em tempo útil. Em contrapartida, os certificados transferidos não estão sujeitos à cache.

Na análise dos certificados envolvidos, pretendeu-se comparar o tamanho dos mesmos para os dois modelos. De igual forma pretendeu-se analisar também a influência do tamanho das chaves. O número típico de certificados no modelo [PKIX](#) também foi tido em conta. Estes factores determinam o tamanho dos dados enviados. O tamanho dos dados tem custos associados de largura de banda e atraso. É desejável diminuir os dados transmitidos em relação ao [PKIX](#) sem degradar a solução.

Escolheram-se alguns sítios, aleatoriamente, que permitem estabelecer sessões [TLS](#) (HTTPS) usando o [PKIX](#). Tipicamente, um servidor envia o certificado do serviço, juntamente com o certificado de uma [CA](#) intermédia que por sua vez é assinado por uma [CA](#) de confiança. Esta última normalmente uma [CA](#) de raiz está presente na lista de certificados de confiança instalados no cliente.

Na tabela 6.5 podem ver-se alguns sítios e respectivo número de certificados transferidos, no momento do estabelecimento da ligação entre servidor e cliente. Estes foram observados através do *Wireshark*. No *browser*, observou-se a informação sobre a cadeia de certificados. Para todos os sítios observados existia uma cadeia composta por três certificados: um certificado do respectivo serviço, um da [CA](#) intermédia e um da [CA](#) de raiz. Os certificados de [CA](#) de raiz para todos os sítios escolhidos já estão incluídos na lista de [CA](#) do *browser*. O *Firefox* já possuía esses certificados na lista de [CA](#). Nestas condições apenas é necessário enviar os restantes dois certificados. Mas noutras situações pode ser necessário enviar o referido certificado. Pode-se concluir que no [PKIX](#) são enviados normalmente dois ou três certificados, mas que a maioria envia apenas dois.

No caso do novo modelo, são enviados sempre apenas dois certificados e a cadeia é composta apenas por esses dois certificados.

Calcularam-se as médias para o [PKIX](#), dos tamanhos dos certificados de serviço e

Tabela 6.6: Tamanho em bytes dos certificados PKIX transmitidos

Sítios	Tamanho dos Certificados (bytes)			
	Serviço	CA intermédia	CA de raiz	Total
www.fe.up.pt	1326	903		2229
www.bes.pt	1296	903	576	2775
www.paypal.com	1514	1550	1249	4313
www.google.com	805	807		1612
caixadirecta.cgd.pt	1090	903		1993
launchpad.net	1130	1093		2223
www.alertpay.com	1344	1184		2528
signin.ebay.com	1676	1512	1249	4437
www.mbnet.pt	1458	1061	606	3125
www.amazon.com	1271	1184		2455
Médias	1291		1056	2769

de CA. Calculou-se ainda a média do tamanho de informação de certificados que foram enviados para cada sessão. Estes valores podem ser observados na tabela 6.6.

No caso do novo modelo, o tamanho dos certificados de serviço e da CA local do nó são bastante inferiores. No caso dos certificados de servidor, em relação ao PKIX, a redução em tamanho é quase para metade. Obteve-se um tamanho de 689 bytes para o certificado do servidor *Web* e um de 744 bytes para o certificado da CA do nó. No total, os certificados enviados no novo modelo representam 1433 bytes de informação transmitida.

O tamanho de um certificado depende do tamanho da chave pública do mesmo e do tamanho da assinatura. O tamanho da assinatura depende do tamanho do módulo da chave pública presente no certificado da CA que o assinou. Foram obtidos os tamanhos das chaves e das assinaturas para analisar a contribuição dos mesmos para o tamanho dos certificados. Na tabela 6.7 pode-se observar os tamanhos das chaves e assinaturas dos certificados PKIX. Os tamanhos obtidos para a chave correspondem ao módulo e expoente, e com os respectivos indicadores, o que significa um tamanho superior a 128 bytes. Por exemplo, no caso do sítio *launchpad.net*, a assinatura do certificado do servidor é realizada por uma chave de 1024 bits. O facto de surgir 138 em vez dos 140 bytes, como noutros casos, explica-se pelo facto do expoente da chave possuir apenas um byte em vez dos três que surgem nos outros certificados.

Pode-se observar que a maior parte dos certificados PKIX possuem chaves de 1024 bits. Como se pode observar todos os certificados dos serviços possuem esse tamanho de chave, que é comum na actualidade. Os certificados das CA, em grande parte, usam também chaves de 1024 bits, contudo, pelos mesmos padrões de segurança, deveriam usar chaves maiores, pelo menos de 2048 bits. Estas considerações estão de acordo com as recomendações NIST (*National Institute of Standards and Technology*) para os tamanhos das chaves em [25]. Pelo facto dos certificados das CA possuírem um período de validade superior ao dos certificados de servidor, alguns, de facto, já possuem chaves de 2048 bits.

Tabela 6.7: Tamanho das chaves e assinaturas presentes nos certificados analisados.

Sítios	Certificados (bytes)					
	Serviço		CA intermédia		CA de raiz	
	Chave	Assin.	Chave	Assin.	Chave	Assin.
www.fe.up.pt	140	128	140	128	140	128
www.bes.pt	140	128	140	128	140	128
www.paypal.com	140	256	270	256	270	256
www.google.com	140	128	140	128	140	128
caixadirecta.cgd.pt	140	128	140	128	140	128
launchpad.net	140	128	138	128	140	128
www.alertpay.com	140	256	270	128	140	128
signin.ebay.com	140	256	270	256	270	256
www.mbnet.pt	140	256	270	128	140	128
www.amazon.com	140	256	270	128	140	128

Nos certificados analisados, os certificados dos servidores não possuíam um período de validade para além do ano 2010.

Para as chaves do novo modelo, pela facilidade de renovação dos certificados, assumiram-se valores bem inferiores. O certificado do servidor tem uma chave de 106 bytes (768 bits) e uma assinatura de 128 bytes. A CA uma chave de 140 bytes (1024 bits) e respectiva assinatura de 128 bytes.

Subtraindo os valores dos tamanhos das chaves e assinaturas obtém-se a dimensão relativa da informação presente nos certificados. Esta informação, para além da relacionada com a validade, sujeito e emissor, também tem associada as várias extensões, necessárias ou recomendadas. Na tabela 6.8 podem-se observar os tamanhos dos certificados PKIX, excluindo as chaves e as respectivas assinaturas. Comparando os valores com os da tabela 6.6, verifica-se que no tamanho total dos certificados, a maior contribuição para o mesmo depende em grande parte do resto da informação, para além das chaves e assinaturas. No modelo PKIX, os certificados possuem inúmeras extensões, principalmente no caso de se tratar de certificados de servidor. Normalmente, os certificados de servidor são emitidos para várias finalidades para além do estabelecimento de sessões SSL/TLS. É comum existirem extensões para indicar o endereço de verificação de revogação.

No novo modelo a subtracção de tamanhos de chaves e assinaturas implica 445 bytes para o certificado do servidor e 476 bytes para o da CA, com um total de 931 bytes. Constitui uma redução considerável em relação aos dados obtidos para o PKIX.

O tamanho de alguns certificados, que foi obtido para o PKIX, deveria ser maior. Segundo as considerações feitas para as chaves, as CA deveriam usar chaves de pelo menos 2048 bits, o que não se verifica em todos os casos no PKIX. Os certificados de servidor possuem o tamanho de chave recomendado, mas a sua assinatura deveria ser maior, face ao módulo de chave da CA ser maior. O aumento de 1024 para 2048 bits nas chaves das CA implica, praticamente, o dobro do tamanho das assinaturas e chaves dos seus certi-

Tabela 6.8: Tamanho dos certificados excluindo as chaves e assinaturas.

Sítios	Tamanho dos Certificados (bytes)			
	Serviço	CA intermédia	CA de raiz	Total
www.fe.up.pt	1058	635		1693
www.bes.pt	1028	635	308	1971
www.paypal.com	1118	1024	851	2993
www.google.com	537	539		1076
caixadirecta.cgd.pt	822	635		1457
launchpad.net	862	827		1689
www.alertpay.com	948	786		1734
signin.ebay.com	1280	986	723	2989
www.mbnet.pt	1062	663	338	2063
www.amazon.com	875	786		1661
Médias	959	695		1933

ficados. Implica também o dobro do tamanho das assinaturas nos certificados assinados. Na tabela 6.9 apresentam-se os valores dos certificados [PKIX](#) considerados anteriormente, considerando as recomendações para os tamanhos das chaves.

Tabela 6.9: Tamanho dos certificados considerando os tamanhos recomendados para as chaves.

Sítios	Tamanho dos Certificados (bytes)			
	Serviço	CA intermédia	CA de raiz	Total
www.fe.up.pt	1454	1161		2615
www.bes.pt	1424	1161	834	3419
www.paypal.com	1514	1550	1377	4441
www.google.com	933	1065		1998
caixadirecta.cgd.pt	1218	1161		2379
launchpad.net	1258	1351		2609
www.alertpay.com	1344	1312		2656
signin.ebay.com	1676	1512	1249	4437
www.mbnet.pt	1458	1189	864	3511
www.amazon.com	1271	1312		2583
Médias	1355	1221		3065

Com base nos tamanhos calculados na tabela 6.9, observa-se que, em média, segundo o mesmo padrão de segurança, a cadeia de certificados transferida em [PKIX](#) é de 2576 bytes ($1355 + 1221$), enquanto que com ScalSec é de 1433 bytes ($689 + 744$), o que corresponde a 55.6 % de uma cadeia de certificados [PKIX](#).

6.3 Conclusões

O novo modelo, em relação ao modelo [PKIX](#), de um modo geral, permite reduzir o processamento no cliente. Por estar orientado para o estabelecimento de sessões [SSL/TLS](#),

recorrendo a certificados melhorados para o modelo, permite a diminuição da informação processada. O novo modelo permite reduzir a transmissão de informação adicional desnecessária. O tamanho dos certificados enviados para o cliente é reduzido em relação ao [PKIX](#), pelo uso de chaves menores, com menor período de validade. A informação presente nos certificados pode ser reduzida, por não serem usadas extensões e outros dados que não são necessários ao modelo.

Capítulo 7

Conclusões

Neste capítulo faz-se a revisão do trabalho desenvolvido. Posteriormente apresentam-se também as conclusões e ou contribuições relevantes sobre o trabalho desenvolvido. Finalmente apresentam-se as possibilidades de trabalho futuro a realizar.

7.1 Revisão do trabalho desenvolvido

Os objectivos foram atingidos com sucesso. Foi realizado o estudo, desenvolvimento, e teste de uma solução de autenticação [SSL/TLS](#), baseado num novo modelo de certificação, o ScalSec. O modelo baseia-se numa arquitectura distribuída, onde o [DNSSec](#) assina e distribui um resumo de uma chave por nó, e o nó por sua vez assina uma chave por serviço ou aplicação. Tanto o nó como a aplicação recorrem a certificados X.509. Este modelo é o resultado dos trabalhos de doutoramento do co-orientador, Jaime Dias, colaborador da Unidade de Telecomunicações e Multimédia do INESC Porto.

Baseada no novo modelo, foi realizada uma descrição da solução. A descrição da solução levou à definição da arquitectura da implementação. No levantamento do estado da arte, e considerando o problema a resolver, realizou-se o estudo e análise das tecnologias envolvidas e relacionadas com a implementação.

No âmbito do [DNS](#) foi realizado um estudo das potencialidades e limitações. Era necessário compreender os conceitos básicos envolvidos. Posteriormente o estudo foi alargado à implementação das extensões de segurança do [DNS](#). Os serviços não foram alvo de desenvolvimento nomeadamente, servidor [HTTP](#), [POP3](#), [IMAP](#) e [SMTP](#). A alteração dos certificados foi alvo de um maior estudo. Ao nível da implementação do cliente, inicialmente, foi necessário realizar um estudo profundo do código do Projecto *Mozilla*, perante o tamanho e complexidade de toda a estrutura e integração das várias bibliotecas. Foi necessário analisar uma quantidade significativa de ficheiros de código e documentação existente.

Posteriormente, a utilização da biblioteca [LWRES](#) adicionou uma maior complexidade à implementação. A documentação desta biblioteca era quase inexistente e sem referências

a casos de implementação. Na análise da implementação foi possível obter resultados interessantes mas uma avaliação rigorosa, envolvendo um maior número de variáveis, não foi possível em tempo útil.

7.2 Contribuições relevantes

As contribuições relevantes são as seguintes:

- O cliente [TLS](#), parte integrante do *Mozilla NSS*, foi alterado de modo a suportar o ScalSec em paralelo com o [PKIX](#). O cliente foi modificado para recorrer a certificados X.509 adaptados e a novos registos por nó, presentes no [DNSSec](#), para autenticar o servidor [TLS](#), suportando sinónimos ([DNS](#)) de forma transparente e segura, sem modificações dos referidos certificados.
- Foi definido um novo registo [DNSSec](#) para suporte de resumos de chaves públicas associadas a nomes ou endereços de nós, *hosts* ou *routers*.
- A semântica dos certificados X.509 foi adaptada de modo a reflectir o novo caminho de certificação, bem como os novos identificadores. Cada nó assume-se como [CA](#) local e emite certificados para cada serviço.
- Foram implementados mecanismos ao nível do nó servidor, que permitem a gestão automática dos respectivos certificados, bem como a actualização dos respectivos registos no [DNS](#), de forma transparente para o utilizador e sem alteração dos processos servidores.

7.3 Trabalho futuro

Ao nível do [DNSSec](#) espera-se que à medida das necessidades, sejam criadas novas e melhores ferramentas para a gestão e manutenção do mesmo. De igual modo, a implementação das extensões de segurança até à raiz, permitiria um maior avanço para uma implementação do novo modelo.

O desenvolvimento nos servidores poderá ser melhorado, passando pela disponibilização de um pacote para instalação.

O desenvolvimento ao nível do cliente poderá ser alargado a outras aplicações para além do *Firefox* e do *Thunderbird*. Este alargamento poderá ser realizado a aplicações já existentes ou a desenvolver futuramente através do código do projecto *Mozilla*.

Anexo A

DNSSec

A.1 Ficheiros das zonas DNS configuradas

A.1.1 db.porto.pt

```
$TTL      604800
$ORIGIN   porto.pt.
@         IN      SOA      ns3.porto.pt. root.porto.pt. (
                                2          ; Serial
                                604800     ; Refresh
                                86400      ; Retry
                                2419200    ; Expire
                                604800 )   ; Negative Cache TTL
@         IN      NS       ns3
          IN      MX       10 w3.porto.pt.
ns3       IN      A        172.16.10.6
www       IN      CNAME    w3
w3        IN      A        172.16.10.6
$INCLUDE  "ksk/Kporto.pt.+005+15852.key"
$INCLUDE  "zsk/Kporto.pt.+005+36467.key"
```

A.1.2 db.10.16.172.in-addr.arpa

```
$TTL      604800
$ORIGIN   10.16.172.in-addr.arpa.
@         IN      SOA      ns3.porto.pt. root.porto.pt. (
                                3          ; Serial
                                604800     ; Refresh
                                86400      ; Retry
                                2419200    ; Expire
                                604800 )   ; Negative Cache TTL
          IN      NS       ns3.porto.pt.
6         IN      PTR      ns3.porto.pt.
```

```

6      IN      PTR    w3.porto.pt.
$INCLUDE "ksk/K10.16.172.in-addr.arpa.+005+38007.key"
$INCLUDE "zsk/K10.16.172.in-addr.arpa.+005+35456.key"

```

A.1.3 db.pt

```

$TTL      604800
$ORIGIN   pt.
@         IN      SOA      ns1.pt. root.pt. (
                                6          ; Serial
                                604800     ; Refresh
                                86400      ; Retry
                                2419200    ; Expire
                                604800 )   ; Negative Cache TTL
; main domain name servers
      IN      NS       ns1.pt.
; A records for name servers above
ns1     IN      A       172.16.10.7
; -----
; sub-domain definitions Porto
$ORIGIN  porto.pt.
; definition of the name servers for the sub-domain
@       IN      NS      ns3.porto.pt.
@       IN      DS      15852 5 1 AB0A781C20EAF36D4453BEDF4012B1A80157FF1
; the next name server points to ns1 above
      IN      NS      ns1.pt.
; address record for sub-domain name server
ns3     IN      A       172.16.10.6
$INCLUDE "zsk/Kpt.+005+18062.key"
$INCLUDE "ksk/Kpt.+005+46155.key"

```

A.1.4 db.16.172.in-addr.arpa

```

$TTL      604800
$ORIGIN   16.172.in-addr.arpa.
@         IN      SOA      ns1.pt. root.pt. (
                                3          ; Serial
                                604800     ; Refresh
                                86400      ; Retry
                                2419200    ; Expire
                                604800 )   ; Negative Cache TTL
; A records for name servers above
      IN      NS      ns1.pt.
; -----
; sub-domain definitions Porto
$ORIGIN   10.16.172.in-addr.arpa.

```

```

; definition of the name servers for the sub-domain
@      IN      NS      ns3.porto.pt.
@      IN      DS      38007 5 1 F74F1FC486AAB81167C56FB63C3F7671611011C9
; the next name server points to ns1 above
;      IN      NS      ns1.pt.
; address record for sub-domain name server
;ns3    IN      A      172.16.10.6
$INCLUDE "zsk/K16.172.in-addr.arpa.+005+18062.key"
$INCLUDE "ksk/K16.172.in-addr.arpa.+005+56418.key"

```

A.2 Exemplo da zona porto.pt assinada

```

porto.pt.      604800  IN  SOA  ns3.porto.pt. root.porto.pt. (
                                2          ; serial
                                604800     ; refresh (1 week)
                                86400      ; retry (1 day)
                                2419200    ; expire (4 weeks)
                                604800     ; minimum (1 week)
                                )
                604800  RRSIG  SOA 5 2 604800 20080706152401 (
                                20080606152401 36467 porto.pt.
                                2QmDTWxXQ3rCV/2o9WvmW2qxdTsktP605sa8
                                ScM25Iob+QEufb55cr0GN1xOG5emWZfmBL3M
                                I9ItxcpDn8qhQoHDyQVpRip/f2ON4c1IN18H
                                BbmVQ5iKJsb5KB7Cx8RW )
                604800  NS     ns3.porto.pt.
                604800  RRSIG  NS 5 2 604800 20080706152401 (
                                20080606152401 36467 porto.pt.
                                pzhckPWYaVwJq01qtVs1VJBR40nQJKEoXkJ0
                                vF1C/QCqDL617fFSLYlHkFFORyHi0tFS2/e1
                                Tccz9jbCp2gW2rb0f7bWa1fu/VpPBF/uvlfm
                                WKXvevwlXkziNpc6Qgq8 )
                604800  MX     10 w3.porto.pt.
                604800  RRSIG  MX 5 2 604800 20080706152401 (
                                20080606152401 36467 porto.pt.
                                opafAo+Cx4pyUkvAiB95iAF6z3jQIYILEf78
                                f16CBv0o6QKXKefpwPQpsIOJg77UPykI81pm
                                5XjKTs4fx3mfVHaOKCK1RquFYlnAoYlBeyXX
                                ibbrG3l8DoPL8aqpzBLS )
                604800  NSEC    ns3.porto.pt. NS SOA MX RRSIG NSEC DNSKEY
                604800  RRSIG  NSEC 5 2 604800 20080706152401 (
                                20080606152401 36467 porto.pt.
                                QqrnfKx/09YQwSe05SD8crVgzIQkbGPg/VXD
                                QFEQMv8nfyVrk22Cv/QEFXs5ppnlpZkwLCpw
                                kknmRU57ZVRiPN3oe/h2MdPiI3MGYU610LZP
                                wmuCgY+3oLQhwG54ZCqf )
                604800  DNSKEY  256 3 5 (

```

```

AQPuJoKXci7ui+ufSvhFIpRZc1Fvx26J2csF
D4+MhYbebPJo2GqQLgzzwbqxpPgFn4Ftqax7
NEOJVngmMtRNGZbEf9jVFloVTloFjl9hYi4I
zMAY1055Hov1tMUXr6MFsQc=
) ; key id = 36467
604800  DNSKEY 257 3 5 (
  AQMOTOG6yohVNnQt82G4MV8Tes6bumUOL8nI
  i8fjixU2XEW38JVH1B6270U9tU1dlf0hERTv
  bAkrdmuX+28Xh5uoudSAC2I5jWFq3HuBI6+b
  JIkT0dalce/x5WLgUF+rtKaGkwJkS0SICV0v
  +b/Sc8/iFcA3zmTqUepuT7G6tCVV18XFXe5j
  EUwx1s1wvjmgrZ10IGomOUudSiqKC2xm0F6T
  JDXr
  ) ; key id = 15852
604800  RRSIG  DNSKEY 5 2 604800 20080706152401 (
  20080606152401 15852 porto.pt.
  AS0vpVbx5GDQ7fE0vj6KcHm5H1+Ni0aEmZTw
  9DBtzUyuAtSNopErkmOvincao0r40vgQQRkH
  DuAF9L1rdm2Di+6RGVhxK63p4H88sYljWWRH
  5eM+cUx3sQngNIGyor+WixK22QBe2GqdM2yx
  cUWKZOV4qbIuy3q/Po1Mfwj/GJsyqbwtN00z
  ErcnH5MS8kl0B0xLMRgCX9VzeSwHDwU2tZgs
  kA== )
604800  RRSIG  DNSKEY 5 2 604800 20080706152401 (
  20080606152401 36467 porto.pt.
  GHP5mtqbRv5Gni2K9Qq/lHobU5h6moCRT9Xj
  CH54c4BN7DNgnjl+mjHo7bHSQWe9upvs4/Jd
  tBcz1TNrMJF88Ts5gRnkmENY4UQ163ABbhmi
  pTC/o/D7roWtsHn8kmpe )
ns3.porto.pt. 604800  IN  A 172.16.10.6
604800  RRSIG  A 5 3 604800 20080706152401 (
  20080606152401 36467 porto.pt.
  jzb/ZmaKbMuX0EV6aBhYh7CtojdlqRN/iqUx
  EBECHrd0wIdypxLYwXP+njNvbSuW6+iZIfE
  T0XnV4sCdaHixsg60ch/rLAFQaZg3S81sCPI
  359BzLyJiqF3AzVILSMN )
w3.porto.pt. 604800  IN  A 172.16.10.6
604800  RRSIG  A 5 3 604800 20080706152401 (
  20080606152401 36467 porto.pt.
  nnZ9z8m5Q8/imTKrAFVP1RrMPGLGVHXSIRKI
  aU/uB684EyYFNVxsH9AxPnuuSuih9xqFJ74W
  tQ+ftAw00NQK6MGEpP5M9oy1f4sUGZO/1033
  hF3Efe+JjhgeM1Bjsey3 )
604800  NSEC  www.porto.pt. A RRSIG NSEC
604800  RRSIG  NSEC 5 3 604800 20080706152401 (
  20080606152401 36467 porto.pt.
  js6+/4EcRxF74EY2caxYja37K9Wnze1XsDGE
  PS0idJgV76FBVGiSoaJET9hqT60/4s+d02Bs

```


Anexo B

Servidores

B.1 Ficheiros de configuração para OpenSSL

B.1.1 Exemplo de ficheiro de configuração para a CA do nó

```
[ ca ]
default_ca = ScalSec_CA
[ ScalSec_CA ]
dir = /home/vmonica/Projects/scripts/ScalSec/ScalSecCA
certs = $dir/certs
database = $dir/index.txt
certificate = $dir/ScalSecCrt.pem
new_certs_dir = $dir/signedcerts
private_key = $dir/private/ScalSecKey.pem
serial = $dir/serial
default_days = 30
default_md = sha1
policy = ScalSec_CA_policy
x509_extensions = ScalSec_service_extensions
[ ScalSec_CA_policy ]
commonName = supplied
stateOrProvinceName = optional
countryName = optional
emailAddress = optional
organizationName = match
organizationalUnitName = optional
[ ScalSec_service_extensions ]
basicConstraints = CA:false
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
nsCertType = server
extendedKeyUsage = serverAuth
[ req ]
default_bits = 1024
```

```

default_keyfile      = $dir/private/ScalSecKey.pem
default_md           = sha1
prompt = no
distinguished_name   = ScalSec_CA_distinguished_name
x509_extensions      = ScalSec_CA_extensions
[ ScalSec_CA_distinguished_name ]
commonName           = 2008-06-03 20:57:00.649319998+00:00
organizationName     = ScalSec
organizationalUnitName = Node
[ ScalSec_CA_extensions ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
basicConstraints      = CA:true, pathlen:0
nsCertType            = sslCA

```

B.1.2 Exemplo de ficheiro de configuração para serviço

```

[ req ]
prompt = no
distinguished_name = server_distinguished_name
[ server_distinguished_name ]
commonName          = __tcp_443
organizationName    = ScalSec
organizationalUnitName = Node

```

B.2 Exemplos de certificados obtidos

B.2.1 Exemplo de um certificado obtido para a CA do nó

Data:

Version: 3 (0x2)

Serial Number:

b4:f6:e6:e6:0e:03:0a:ac

Signature Algorithm: sha1WithRSAEncryption

Issuer: CN=2008-06-03 20:57:00.649319998+00:00, O=ScalSec, OU=Node

Validity

Not Before: Jun 3 20:57:01 2008 GMT

Not After : Jul 4 20:57:01 2008 GMT

Subject: CN=2008-06-03 20:57:00.649319998+00:00, O=ScalSec, OU=Node

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:cd:59:5d:a5:9c:77:63:14:21:bc:9d:59:0b:5c:

4d:13:6f:9b:63:d2:cc:07:26:13:01:41:4c:b1:cf:

4a:96:f3:d9:5b:ca:3a:cf:de:f5:57:00:30:cf:73:

```

1d:d8:46:86:c2:53:da:c0:de:5a:0d:ad:b4:86:b2:
59:00:08:20:92:db:ba:61:83:f7:73:10:55:5f:ff:
d2:1b:a4:5d:62:85:d4:34:9e:1e:d5:06:ac:05:4d:
f6:19:1d:a8:6e:e7:38:9f:a8:cd:98:11:f1:11:63:
08:75:d7:01:45:5e:87:e0:35:6b:bd:6b:51:3b:1c:
27:c6:7e:8c:b8:26:88:20:6f
Exponent: 65537 (0x10001)
X509v3 extensions:
  X509v3 Subject Key Identifier:
    4B:AC:6A:36:AB:E2:A2:B7:B8:2F:84:B2:A6:3C:51:42:67:22:3E:58
  X509v3 Authority Key Identifier:
    keyid:4B:AC:6A:36:AB:E2:A2:B7:B8:2F:84:B2:A6:3C:51:42:67:22:3E:58
    DirName:/CN=2008-06-03 20:57:00.649319998+00:00/0=ScalSec/OU=Node
    serial:B4:F6:E6:E6:0E:03:0A:AC
  X509v3 Basic Constraints:
    CA:TRUE, pathlen:0
  Netscape Cert Type:
    SSL CA
Signature Algorithm: sha1WithRSAEncryption
53:b7:24:47:56:33:76:58:9a:23:b4:00:44:75:23:f5:07:06:
67:02:0b:ff:02:99:f5:73:28:5c:64:37:1e:50:8d:12:84:25:
70:99:9d:28:dc:8d:03:e1:2b:05:03:b2:7e:26:55:ed:ef:de:
b1:bc:46:82:49:32:a4:2d:45:c6:92:b7:a2:34:95:9e:fb:77:
dd:22:20:a1:86:05:0d:dd:ed:94:55:0a:39:22:59:b8:d6:21:
99:a4:1d:c0:59:74:65:8b:a5:35:73:3f:0e:d6:92:6a:2a:73:
58:d2:4e:2b:97:c9:05:54:51:d5:f7:ca:ea:20:2d:cf:b0:a8:
4a:46

```

B.2.2 Exemplo de um certificado obtido para um serviço

```

Data:
  Version: 3 (0x2)
  Serial Number: 1 (0x1)
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: CN=2008-06-03 20:57:00.649319998+00:00, O=ScalSec, OU=Node
  Validity
    Not Before: Jun  3 20:57:01 2008 GMT
    Not After : Jun  4 20:57:01 2008 GMT
  Subject: CN=__tcp_443, O=ScalSec, OU=Node
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (768 bit)
    Modulus (768 bit):
      00:e7:59:97:f1:d3:a0:bd:06:41:43:f2:7d:bd:03:
      61:77:74:5b:53:0c:c2:af:7e:29:48:db:7c:9f:40:
      83:e2:9b:f8:7e:a6:44:3d:33:d4:37:4f:44:0b:75:
      02:d3:65:4c:16:75:cd:a7:ae:a4:e5:37:c4:e4:29:

```

```
        ba:14:c3:fd:e6:ce:c6:d4:d5:e4:15:38:aa:61:aa:
        b4:6c:10:56:b9:a7:d9:dd:e6:9a:c6:5d:21:fb:1f:
        6b:eb:d7:fa:93:cc:eb
    Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    X509v3 Subject Key Identifier:
        CF:7D:1D:D6:8C:3C:51:CA:A5:01:B9:25:F1:48:84:F9:33:B3:F9:E0
    X509v3 Authority Key Identifier:
        keyid:4B:AC:6A:36:AB:E2:A2:B7:B8:2F:84:B2:A6:3C:51:42:67:22:3E:58
        DirName:/CN=2008-06-03 20:57:00.649319998+00:00/0=ScalSec/OU=Node
        serial:B4:F6:E6:E6:0E:03:0A:AC
    Netscape Cert Type:
        SSL Server
    X509v3 Extended Key Usage:
        TLS Web Server Authentication
Signature Algorithm: sha1WithRSAEncryption
60:bc:4f:0b:96:8a:39:e3:0b:77:ac:50:12:46:ba:e5:35:f4:
50:25:d5:a2:b6:60:2e:f7:3f:ff:0f:4a:d1:8c:a1:1f:79:d8:
57:aa:f5:38:fc:3a:06:0c:32:1f:44:65:cf:68:8e:47:11:d6:
3f:36:5c:d9:9b:17:18:8b:58:ea:5e:c1:76:17:13:27:65:b1:
24:c9:9d:b1:eb:a4:e8:be:e9:d8:57:6e:b4:78:21:67:e9:ae:
1b:6c:6a:37:3f:1d:38:32:70:a1:98:d9:17:38:f6:f9:74:a6:
c3:49:eb:1f:df:a2:56:8f:e3:57:52:15:1a:cd:f4:ac:3f:6f:
ba:6e
```

Anexo C

Cliente

C.1 Configuração do LWRES

```
options{
    directory "/var/cache/bind";
    dnssec-enable yes;
    dnssec-validation yes;
    forward only;
    listen-on-v6{any;};
    forwarders {
        172.16.10.7;
    };
};
trusted-keys {
    "pt." 256 3 5 "AQ0v3wVb2k6LsSz/56jL60kCQ2eke6YcFksI7cagUiBA/AwuJyb8X0Ii
        g09EcZ1SSAulNB6oQLNidZviX5+1D8G4LWkLH+bwYrhmKILn5qc6WuTA
        tIKHwgtAH+x3e+Tt8ps=";
};
lwres{};
server 172.16.10.7{
    keys{valter.porto.pt. };
};
key valter.porto.pt.{
    algorithm hmac-md5;
    secret "Me9gkynH0t1JViC6EfFpNA==";
};
```

C.2 Código alterado

C.2.1 Alterações no ficheiro sslauth.c

SECStatus

```

SSL_AuthCertificate(void *arg, PRFileDesc *fd, PRBool checkSig,
                    PRBool isServer){
    SECSStatus      rv;
    CERTCertDBHandle * handle;
    sslSocket *      ss;
    SECCertUsage      certUsage;
    const char *      hostname = NULL;
    unsigned short ssport;

    ss = ssl_FindSocket(fd);
    PORT_Assert(ss != NULL);
    if (!ss) {
        return SECFailure;
    }
    handle = (CERTCertDBHandle *)arg;

    /* this may seem backwards, but isn't. */
    certUsage = isServer ? certUsageSSLClient : certUsageSSLServer;

    char* SOrgName=CERT_GetOrgName(&ss->sec.peerCert->issuer);
    if(PORT_Strcmp(SOrgName,"ScalSec")==0){
        hostname=ss->url;
        ssport=PR_ntohs(ss->sec.ci.port);
        rv=CERT_VerifyScalSecNow(handle, ss->sec.peerCert,
                                checkSig,
                                certUsage, ss->pkcs11PinArg,
                                hostname, ssport);
    }else{
rv = CERT_VerifyCertNow(handle, ss->sec.peerCert, checkSig, certUsage,
                        ss->pkcs11PinArg);
    }
    if ( rv != SECSuccess || isServer )
        return rv;
    /* cert is OK.  This is the client side of an SSL connection.
     * Now check the name field in the cert against the desired hostname.
     * NB: This is our only defense against Man-In-The-Middle
     */ (MITM) attacks!
        if(PORT_Strcmp(SOrgName,"ScalSec")!=0){
hostname = ss->url;
if (hostname && hostname[0])
    rv = CERT_VerifyCertName(ss->sec.peerCert, hostname);
else
    rv = SECFailure;
if (rv != SECSuccess)
    PORT_SetError(SSL_ERROR_BAD_CERT_DOMAIN);
    }
    PORT_Free(SOrgName);
    return rv;
}

```

```
}

```

C.2.2 Alterações no ficheiro certvfy.c

```
static SECStatus
cert_ScalSec(CERTCertDBHandle *handle, CERTCertificate *cert,
             PRBool checkSig, PRBool* sigerror, SECCertUsage certUsage,
             int64 t, void *winctx, CERTVerifyLog *log,
             PRBool* revoked, char* url){

    SECTrustType trustType;
    CERTBasicConstraints basicConstraint;
    unsigned int caCertType;
    unsigned int requiredCAKeyUsage;
    unsigned int requiredFlags;
    int currentPathLen=0;
    int pathLengthLimit=CERT_UNLIMITED_PATH_CONSTRAINT;
    PRBool isFortezzaV1=PR_FALSE;
    PRBool isca;
    SECStatus rv;
    SECStatus rvFinal = SECSuccess;
    CERTCertificate *subjectCert=NULL;
    CERTCertificate *issuerCert=NULL;
    unsigned int rdclass=1;
    unsigned int rdtype=120;
    unsigned int flags=0;
    struct rrsetinfo *res=NULL;
    int result;
    int i;
    unsigned int algorithm;
    unsigned int digest_type;
    unsigned char PublicKeyHash[SHA1_LENGTH];
    SECItem hashpubkeycert;
    SECItem digestpubdns;
    PRNetAddr netAddr;
    char* host=NULL;
    char hostrev[128]="\0";
    const char* hex="0123456789abcdef";
    int h=0;

    if (CERT_KeyUsageAndTypeForCertUsage(certUsage, PR_TRUE,
                                         &requiredCAKeyUsage,
                                         &caCertType) != SECSuccess){
        PORT_Assert(0);
        EXIT_IF_NOT_LOGGING(log);
        requiredCAKeyUsage=0;
        caCertType=0;
    }
}
```

```

}
switch ( certUsage ) {
    case certUsageSSLServer:
        if ( CERT_TrustFlagsForCACertUsage(certUsage,
                                            &requiredFlags,
                                            &trustType)
            != SECSuccess ) {

            PORT_Assert(0);
            EXIT_IF_NOT_LOGGING(log);
            requiredFlags = 0;
            trustType = trustSSL;
        }
        break;
    default:
        PORT_Assert(0);
        EXIT_IF_NOT_LOGGING(log);
        requiredFlags = 0;
        trustType = trustSSL;
/* This used to be 0, but we need something
 * that matches the enumeration type.*/
        caCertType = 0;
}

subjectCert=CERT_DupCertificate(cert);
if(subjectCert==NULL){
    goto loser;
}
if(subjectCert->options.bits.hasUnsupportedCriticalExt){
    PORT_SetError(SEC_ERROR_UNKNOWN_CRITICAL_EXTENSION);
    LOG_ERROR_OR_EXIT(log,subjectCert,0,0);
}
issuerCert=CERT_FindCertIssuer(subjectCert,t,certUsage);
if(!issuerCert){
    PORT_SetError(SEC_ERROR_UNKNOWN_ISSUER);
    LOG_ERROR(log,subjectCert,0,0);
    goto loser;
}
if(checkSig){
    rv=CERT_VerifySignedData(&subjectCert->signatureWrap,
                            issuerCert,t,wincx);

    if(sigerror){
        *sigerror=PR_TRUE;
    }
    if(rv!=SECSuccess){
        if(PORT_GetError()==SEC_ERROR_EXPIRED_CERTIFICATE){
            PORT_SetError(SEC_ERROR_EXPIRED_ISSUER_CERTIFICATE);
            LOG_ERROR_OR_EXIT(log,issuerCert,1,0);
        }else{
            PORT_SetError(SEC_ERROR_BAD_SIGNATURE);

```

```

        LOG_ERROR_OR_EXIT(log,subjectCert,0,0);
    }
}
}
rv=CERT_FinBasicConstraintExten(issuerCert,&basicConstraint);
if(rv!=SECSuccess){
    if(PORT_GetError()!=SEC_ERROR_EXTENSION_NOT_FOUND){
        LOG_ERROR_OR_EXIT(log,issuerCert,1,0);
    }
    pathLengthLimit=CERT_UNLIMITED_PATH_CONSTRAINT;
    isca=isFortezzaV1;
}else{
    if(basicConstraint.isCA==PR_FALSE){
        PORT_SetError(SEC_ERROR_CA_CERT_INVALID);
        LOG_ERROR_OR_EXIT(log,issuerCert,1,0);
    }
    pathLengthLimit= basicConstraint.pathLenConstraint;
    isca=PR_TRUE;
}
if(pathLengthLimit >= 0 && currentPathLen > pathLengthLimit){
    PORT_SetError(SEC_ERROR_PATH_LEN_CONSTRAINT_INVALID);
    LOG_ERROR_OR_EXIT(log,issuerCert,1,pathLengthLimit);
}
if(!isca || (issuerCert->nsCertType & NS_CERT_TYPE_CA)){
    isca=(issuerCert->nsCertType
        & caCertType) ? PR_TRUE:PR_FALSE;
}
if(!isca){
    PORT_SetError(SEC_ERROR_CA_CERT_INVALID);
    LOG_ERROR_OR_EXIT(log,issuerCert,1,0);
}
if(CERT_CheckKeyUsage(issuerCert,requiredCAKeyUsage)!=SECSuccess){
    PORT_SetError(SEC_ERROR_INADEQUATE_KEY_USAGE);
    LOG_ERROR_OR_EXIT(log,issuerCert,1,requiredCAKeyUsage);
}
/* ***** */
if(PR_StringToNetAddr(url,&netAddr)==PR_SUCCESS){
    unsigned char* ip;
    unsigned int lownibble;
    unsigned int highnibble;

    switch(netAddr.raw.family){
        case PR_AF_INET:
            ip = (unsigned char *) &netAddr.inet.ip;
            sprintf(hostrev,"%d.%d.%d.%d.in-addr.arpa",
                ip[3],ip[2],ip[1],ip[0]);
            break;
        case PR_AF_INET6:

```

```

        ip = (unsigned char*) &netAddr.ipv6.ip;
        for(h=15; h>=0; h--){

            lownibble= (ip[h] & 0x0F);
            highnibble= (ip[h] & 0xF0) >> 4;
            PORT_Strncat(hostrev,hex+lownibble,1);
            PORT_Strcat(hostrev,".");
            PORT_Strncat(hostrev,hex+highnibble,1);
            PORT_Strcat(hostrev,".");
        }
        PORT_Strcat(hostrev,"ip6.arpa");
        break;
    }
    host=hostrev;
}else{
    host=url;
}
result=getrrsetbyname(host,rdclass,rdtype,flags,&res);
switch(result){
    case ERRSET_SUCCESS:
        if(!(res->rri_flags & RRSET_VALIDATED)){
            PORT_SetError(SEC_ERROR_UNTRUSTED_ISSUER);
            LOG_ERROR_OR_EXIT(log,issuerCert,1,0);
        }
        KeyType certpkeyalg = CERT_GetCertKeyType(
            &issuerCert->subjectPublicKeyInfo);
        for(i=0; i< res->rri_nrdatas; i++){
            unsigned char *rdata=res->rri_rdatas[i].rdi_data;
            unsigned int rdata_len=res->rri_rdatas[i].rdi_length;
            unsigned int digestlen=0;
            KeyType rrpkeyalg = 0;
            SECObjectIdTag hash_type;

            algorithm = (unsigned int) rdata[0];
            switch(algorithm){
                case 0:
                    rrpkeyalg=nullKey;
                    break;
                case 1:
                    rrpkeyalg=rsaKey;
                    break;
                case 2:
                    rrpkeyalg=dsaKey;
                case 3:
                    rrpkeyalg=ecKey;
                default:
                    rrpkeyalg=nullKey;
                    break;
            }
        }
    }
}

```

```

    }
    if(rrpkeyalg==certpkeyalg){
        digest_type= (unsigned int) rdata[1];
        digestlen=rdata_len - 2;
        digestpubdns.data =(unsigned char*)PORT_Alloc(digestlen);
        digestpubdns.len = digestlen;
        PORT_Memcpy(digestpubdns.data,rdata + 2, digestlen);

        switch(digest_type){
            case 0:
                break;
            case 1:
                hash_type=SEC_OID_SHA1;
                break;
            default:
                break;
        }

        PK11_HashBuf(hash_type,PublicKeyHash,
                    issuerCert->derPublicKey.data,
                    issuerCert->derPublicKey.len);
        hashpubkeycert.data=PublicKeyHash;
        hashpubkeycert.len=SHA1_LENGTH;
        if(PR_TRUE==SECITEM_ItemsAreEqual(&hashpubkeycert,
                                           &digestpubdns)){
rv=rvFinal;

            goto done;
        }

        if(digestpubdns.data){
            PORT_Free(digestpubdns.data);
        }
    }

    break;
case ERRSET_NOMEMORY:
    PORT_SetError(SEC_ERROR_NO_MEMORY);
    LOG_ERROR_OR_EXIT(log,issuerCert,1,0);
    break;
case ERRSET_FAIL:
    PORT_SetError(SEC_ERROR_LIBRARY_FAILURE);
    LOG_ERROR_OR_EXIT(log,issuerCert,1,0);
    break;
case ERRSET_INVALID:
    PORT_SetError(SEC_ERROR_INVALID_ARGS);
    LOG_ERROR_OR_EXIT(log,issuerCert,1,0);
    break;
case ERRSET_NONAME:
    PORT_SetError(SEC_ERROR_UNKNOWN_ISSUER);

```

```

        LOG_ERROR_OR_EXIT(log, issuerCert, 1, 0);
        break;
    case ERRSET_NODATA:
        PORT_SetError(SEC_ERROR_UNKNOWN_ISSUER);
        LOG_ERROR_OR_EXIT(log, issuerCert, 1, 0);
        break;
}

PORT_SetError(SEC_ERROR_UNKNOWN_ISSUER);
LOG_ERROR(log, subjectCert, 0, 0);

loser:
    rv=SECFailure;
done:
    if(issuerCert){
        CERT_DestroyCertificate(issuerCert);
    }
    if(subjectCert){
        CERT_DestroyCertificate(subjectCert);
    }
    if(res){
        freerrset(res);
    }
    if(digestpubdns.data){
        PORT_Free(digestpubdns.data);
    }
    return rv;
}

SECStatus
CERT_ScalSec(CERTCertDBHandle *handle, CERTCertificate *cert,
             PRBool checkSig, SECCertificateUsage certUsage,
             int64 t, void *wincx, CERTVerifyLog *log, char* url)
{
    return cert_ScalSec(handle, cert, checkSig, NULL, certUsage,
                        t, wincx, log, NULL, url);
}

SECStatus
CERT_VerifyScalSec(CERTCertDBHandle *handle, CERTCertificate *cert,
                  PRBool checkSig, SECCertUsage certUsage, int64 t,
                  void *wincx, CERTVerifyLog *log, char* url,
                  unsigned short port){
    SECStatus rv;
    unsigned int requiredKeyUsage;
    unsigned int requiredCertType;
    unsigned int certType;
    PRBool allowOverride;
    SECCertTimeValidity validity;

```

```

char* saveptr=NULL;
char* SCommonName=NULL;
char* prtclcert=NULL;
char* portcert=NULL;
unsigned short portcertn;

/* make sure that the cert is valid at time t */
allowOverride = (PRBool)((certUsage == certUsageSSLServer) ||
                          (certUsage == certUsageSSLServerWithStepUp));
validity = CERT_CheckCertValidTimes(cert, t, allowOverride);
if ( validity != secCertTimeValid ) {
    LOG_ERROR_OR_EXIT(log,cert,0,validity);
}
/* check key usage and netscape cert type */
cert_GetCertType(cert);
certType = cert->nsCertType;
switch ( certUsage ) {
    case certUsageSSLServer:
    case certUsageSSLServerWithStepUp:
        rv = CERT_KeyUsageAndTypeForCertUsage(certUsage, PR_FALSE,
                                                &requiredKeyUsage,
                                                &requiredCertType);

        if ( rv != SECSuccess ) {
            PORT_Assert(0);
            EXIT_IF_NOT_LOGGING(log);
            requiredKeyUsage = 0;
            requiredCertType = 0;
        }
        break;
    default:
        PORT_Assert(0);
        EXIT_IF_NOT_LOGGING(log);
        requiredKeyUsage = 0;
        requiredCertType = 0;
}
if ( CERT_CheckKeyUsage(cert, requiredKeyUsage) != SECSuccess ) {
    PORT_SetError(SEC_ERROR_INADEQUATE_KEY_USAGE);
    LOG_ERROR_OR_EXIT(log,cert,0,requiredKeyUsage);
}
if ( !( certType & requiredCertType ) ) {
    PORT_SetError(SEC_ERROR_INADEQUATE_CERT_TYPE);
    LOG_ERROR_OR_EXIT(log,cert,0,requiredCertType);
}

/**/
SCommonName=CERT_GetCommonName(&cert->subject);
prtclcert=strtok_r(SCommonName,"",&saveptr);
portcert=strtok_r(NULL,"",&saveptr);

```

```

    portcertn=(unsigned short)PORT_Atoi(portcert);
    if(port!=portcertn){
        PORT_SetError(SEC_ERROR_UNTRUSTED_CERT);
        rv=SECFailure;
        return rv;
    }
    PORT_Free(SCommonName);
    rv=CERT_ScalSec(handle,cert,checkSig,certUsage,t,wincx,log,url);
    if (rv != SECSuccess) {
        EXIT_IF_NOT_LOGGING(log);
    }
    return(SECSuccess);
loser:
    rv = SECFailure;
    return(rv);
}

SECStatus
CERT_VerifyScalSecNow(CERTCertDBHandle *handle,CERTCertificate *cert,
                      PRBool checkSig,SECCertUsage certUsage,
                      void *wincx,char* url, unsigned short port)
{
    return(CERT_VerifyScalSec(handle,cert,checkSig,
                              certUsage,PR_Now(),wincx,NULL,url,port));
}

```

Referências

- [1] P. Burkholder. SSL Man-in-the-Middle Attacks. V2. 0 [Z]. SANS Institute, 2002. <http://www.sans.org/rr/papers/60/480.pdf>. Acesso em 30/Junho/2008.
- [2] Openssl: The open source toolkit for ssl/tls. <http://www.openssl.org>. Acesso em 30/Junho/2008.
- [3] C. Nwosisi, R. Washington, R. DiGiorgio, and A. Nieto. The Secure Sockets Layer (SSL) Protocol.
- [4] B. Schneier and C. Ellison. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.
- [5] C. Adams and M. Just. PKI: Ten Years Later. *the 3rd Annual PKI R&D Workshop, NIST*, 2004.
- [6] M.S. Bargh J.F. Zandbelt, R.J. Hulsebosch and R. Arends. Trusted directory services for secure internet connectivity: Transport layer security using dnssec. *Electronic Notes in Theoretical Computer Science*, Volume 197:Pages 91–103, February 2008.
- [7] R. Dhamija, JD Tygar, and M. Hearst. Why phishing works. *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590, 2006.
- [8] E. Gerck. Overview of Certification Systems: X. 509, PKIX, CA, PGP & SKIP. *The Bell*, 1(3):8, 2000.
- [9] D.E. Clarke. *SPKI/SDSI HTTP Server/Certificate Chain Discovery in SPKI/SDSI*. PhD thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2001. <http://groups.csail.mit.edu/cis/theses/clarke-masters.pdf>. Acesso em 30/Junho/2008.
- [10] J.E. Elie. *Certificate Discovery Using SPKI/SDSI 2.0 Certificates*. PhD thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 1998. <http://theory.csail.mit.edu/~cis/theses/elien-masters.pdf>. Acesso em 30/Junho/2008.
- [11] C. Fetzer and T. Jim. Incentives and Disincentives for DNSSEC Deployment. 2004. <http://www.research.att.com/~trevor/papers/dnssec-incentives.pdf>. Acesso em 30/Junho/2008.
- [12] J. Gustafsson. *Using the Domain Name System and its Security Extensions as a Public Key Infrastructure*.
- [13] A. Ozment, S.E. Schechter, and R. Dhamija. Web Sites Should Not Need to Rely On Users to Secure Communications. <http://www.cl.cam.ac.uk/~jo262/papers/24-ozment-dont-rely.pdf>. Acesso em 30/Junho/2008.

- [14] J. Schlyter. Storing application public keys in the DNS. *draft-schlyter-appkey-02, IETF, Internet Draft, February*, 2002.
- [15] R. Chaparro, P. Greenwood, and B. Barán. Alternativa de Infraestructura de Clave Pública Basada en el uso de DNSSEC. <http://www.cnc.una.py/invest/paper2/chaCLEI.pdf>. Acesso em 30/Junho/2008.
- [16] D. Massey and S. Rose. Limiting the scope of the KEY resource record (RR). RFC 3445, Internet Engineering Task Force, December 2002.
- [17] D.F. Berger. *A Scalable Architecture for Public Key Distribution Acting in Concert with Secure DNS*. PhD thesis, UNIVERSITY OF CALIFORNIA, 2004.
- [18] JP Jones, DF Berger, and CV Ravishankar. Layering Public Key Distribution Over Secure DNS using Authenticated Delegation. *Computer Security Applications Conference, 21st Annual*, pages 409–418, 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1565268. Acesso em 30/Junho/2008.
- [19] J. Schlyter. DNS as X.509 PKIX certificate storage. *draft-schlyter-pkix-dns-02, February*, 2002.
- [20] J. Schlyter and W. Griffin. Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints. RFC 4255 (Proposed Standard), January 2006.
- [21] ISC BIND. BIND 9 Administrator Reference Manual, 9. <http://www.isc.org/sw/bind/arm95/Bv9ARM.pdf>. Acesso em 30/Junho/2008.
- [22] O. Kolkman. DNSSEC HOWTO, a tutorial in disguise. http://www.nlnetlabs.nl/dnssec_howto/dnssec_howto.pdf. Acesso em 30/Junho/2008.
- [23] R. Aitchison. *Pro Dns And Bind*. Apress, 2005.
- [24] L. Dostalek and A. Kabelova. *Dns in Action: A Detailed And Practical Guide to Dns Implementation, Configuration, And Administration*. 2006.
- [25] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management—Part 1: General (Revised). *NIST Special Publication 800-57*, page 66, 2006.