FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Autenticação de redes Wi-Fi recorrendo ao DNSSEC

Pedro Maia

Tese submetida no âmbito do

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Major de Telecomunicações

Orientador: Manuel Ricardo (Prof Dr.)

Co-orientador: Jaime Dias (Mestre.)

Junho de 2009

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES 2008/2009

A Dissertação intitulada

"AUTENTICAÇÃO DE REDES WI-FI RECORRENDO AO DNSSEC"

foi aprovada em provas realizadas em 16/Julho/2009

o júri

Presidente Professor Doutor João Francisco Cordeiro de Oliveira Barros
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Francisco Manuel Marques Fontes
Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro;

Professor Doutor Manuel Alberto Pereira Ricardo
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

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

Autor - PEDRO MIGUEL MOREIRA MAIA

Faculdade de Engenharia da Universidade do Porto

Resumo

Nas rede Wi-Fi identificou-se como sendo um problema a inexistência de regras, que impeçam duas redes de terem o mesmo *Service Set IDentifier* (SSID) e possuirem servidores de autenticação distintos.

A não instalação de certificados de raiz de *Certification Authority* (CA) pelo utilizador e consequente má prática na configuração do *supplicant*, no sentido da não verificação dos mesmos, constitui outro dos problemas, facto que origina uma autenticação insegura possibilitando assim diversos tipos de ataques à estação.

Tendo como base os problemas identificados deu-se início ao trabalho, que tem como tema, "Autenticação de redes Wi-Fi recorrendo ao DNSSEC". Neste sentido, tem-se como objectivo delineador de todo o trabalho a definição e implementação de uma solução de segurança que permita às estações 802.11 identificar correctamente redes Wi-Fi baseadas no servidor de autenticação, de forma a evitar redes forjadas sem que o utilizador tenha de introduzir mais informação para além do SSID.

O presente trabalho é constituído por duas partes distintas. Numa primeira parte procedeu-se à alteração e configuração do *supplicant Wi-Fi Protected Access* (WPA) para este se autenticar perante redes Wi-Fi com um *SSID global*. Na segunda parte, procedeu-se às alterações necessárias para a autenticação ScalSec, a qual permite verificar a validade do certificado através do *Domain Name System Security Extensions* (DNSSEC). As soluções não podiam ser radicais, pois devem permitir ao utilizador, de forma transparente, os outros tipos de autenticação.

Dos vários protocolos de autenticação existentes deu-se, neste caso, atenção especial ao EAP-PEAP o qual utiliza durante o seu processo de autenticação certificados que garantem ao utilizador a validade do servidor, assegurando assim a confidencialidade da transmissão de dados. O problema deste tipo de solução assenta na necessidade de ter um certificado de raiz de CA o qual implica custos, uma vez que, ou é assinado por uma entidade validada e fidedigna, ou tem que ser instalado localmente nos clientes.

Por fim e de modo a verificar este novo conceito de autenticação, configurou-se um conjunto de máquinas que permite demonstrar e validar esta solução. Foi desenvolvido um conjunto de *scripts* e programas que facilitam a configuração deste processo tornando-o transparente para o administrador da rede.

Abstract

In Wi-Fi networks was identified the lack of rules as being a problem, actually there is no rule that prevents the network administrator from setting the same *Service Set IDentifier* (SSID) that the neighborhood network has already set, well maybe only the rule good sense. Other of the problems that was identified, consisted in bad administrative measures by the station users, that prefer to don't install *Certification Authority* (CA) certificate, and prefer instead to ignore the verification of the CA certificate. This fact alone constitute an insecure autentication, and makes possible to the station to suffer various kinds of attacks.

Was based on the problems above mentioned that this work was initiated, it has the theme "Autenticação de redes Wi-Fi recorrendo ao DNSSEC". In this sense, the main guideline objective of this work is to implement and design a security solution that allows stations 802.11 to correctly identify Wi-Fi networks with authentication servers in order to prevent attacks from forged networks without the user having to enter any information beyond the SSID.

The present work, consists in two parts, in the first part the *Wi-Fi Protected Access* (WPA) *supplicant* was altered and configured to authenticate in Wi-Fi networks with a *global SSID*. In the second part, the needed changes were performed so that a ScalSec authentication could occur, this will allow to verify the validity of the CA certificates using the *Domain Name System Security Extensions* (DNSSEC). The solutions were not radical, it must allow other authentication protocols occur, in a clean and safe manner.

In order to achieve the second objective, various authentication protocols could had be chosen, but in this case, the choosed method was the EAP-PEAP, which uses during the authentication process, certificates, this guarantees the server validity to the user and also ensures the confidentiality of the transmited data.

The problem with the currents solutions is the need of having a CA root certificate. Or the CA root certificate, is signed and validated by a trusted entities, which has its costs, or it has to be locally installed on all customers who want to use the service.

Finally, in order to verify this new authentication approach a set of machines was setuped wich allowed to demonstrate and validate this solution. It was developed a set of scripts and programs that facilitate the configuration process, making it transparent to the network administrator.

Prefácio

À minha família e à minha namorada um obrigado sentido pela paciência e incentivo.

Com a elaboração desta dissertação chegam ao fim meses de trabalho, dedicação e empenho referente ao tema "Autenticação de redes Wi-Fi recorrendo ao DNSSEC". Foi um trabalho árduo, caracterizado por avanços e recuos, pois embora a segurança em redes assuma nos dias de hoje um papel de extrema importância, é ainda uma área "ignorada". No entanto, as dificuldades foram superadas.

Aproveito aqui para fazer os meus agradecimentos ao Instituto de Engenharia de Sistemas e Computadores (INESC), por me deixar utilizar as instalações e equipamentos necessários para a execução deste trabalho.

Não posso deixar de referir a importante ajuda e paciência do Eng. Jaime Dias na execução deste projecto, assim como do Professor Doutor Manuel Ricardo e todas as pessoas que trabalham na Unidade de Telecomunicações e Multimédia (UTM) e WiMobNet.

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Siglas

AAA	*Authentication, Authorization, Accounting*.
AP	*Acess Point*.
AS	*Autentication Server*.
BIND	*Berkeley Internet Name Domain*.
BSS	*Basic Service Set*.
CA	*Certification Authority*.
DDNS	*Dynamic Domain Name Server*.
DNS	*Domain Name System*.
DNSSEC	*Domain Name System Security Extensions*.
DoS	*Denial of Service*.
DS	*Delegation Signer*.
EAP	*Extensible Authentication Protocol*.
ESS	*Extended Service Set*.
FQDN	*Fully Qualified Domain Name*.
IEEE	*Institute of Electrical and Electronics Engineer*.
IETF	*Internet Engineering Task Force*.
ISC	*Internet Systems Consortium*.
ISP	*Internet Service Provider*.
ITU	*International Telecommunications Union*.
KSK	*Key Signing Key*.
LWRESD	*Lightweight Resolver Daemon*.
NAS	*Network Access Server*.
OCSP	*Online Certificate Status Protocol*.

PEAP *Protected Extensible Authentication Protocol*.
PKI *Public Key Infrastructure*.
PPP *Point-to-Point Protocol*.

RADIUS *Remote Autentication Dial In User Service*.
RFC *Request for Comments*.
RR *Ressource Record*.

SEP *Secure Entry Point*.
SSID *Service Set IDentifier*.
SSL *Secure Socket Layer*.
STCP *Secure Transmission Control Protocol*.

TCP *Transmission Control Protocol*.
TKIP *Temporal Key Integrity Protocol*.
TLS *Transport Layer Security*.
TSIG *Transaction SIGnature*.
TTL *Time To Live*.

UDP *User Datagram Protocol*.

WEP *Wired Equivalent Privacy*.
WPA *Wi-Fi Protected Access*.
WPA2 *Wi-Fi Protected Access 2*.

ZSK *Zone Signing Key*.

Capítulo 1

Introdução

1.1 Enquadramento

Com o crescente desenvolvimento das redes Wi-Fi verifica-se, paralelamente, um aumento de exigências a nível da mobilidade, bem como da segurança dessas redes. Com o intuito de promover cada vez mais a segurança, vários protocolos de autenticação foram desenvolvidos e várias normas do *Institute of Electrical and Electronics Engineer* (IEEE) foram revogadas.

Relativamente à norma 802.11 que rege as redes Wi-Fi, esta permite que duas redes vizinhas tenham o mesmo *Service Set IDentifier* (SSID), ou seja, não há qualquer regra na norma que proíba tal facto. Analisando esta situação pode imaginar-se o aparecimento de diferentes problemas, nomeadamente, o *supplicant* pode autenticar-se no *Acess Point* (AP) que não pertence à rede à qual se pretendia ligar, colocando deste modo a estação perante uma grave falha de segurança. Num caso de *roaming* entre APs a interface de rede pode autenticar-se num AP incorrecto e provavelmente falso, devido ao sinal ser mais forte, colocando desta forma a estação em perigo.

As redes Wi-Fi do ponto de vista de segurança não são perfeitas, no entanto têm vindo a melhorar com a introdução e implementação da norma 802.11i. Esta norma permite em redes de grande escala o uso de um servidor de autenticação *Autentication Server* (AS), através do protocolo 802.1X. O protocolo de autenticação mais usado é o *Remote Autentication Dial In User Service* (RADIUS) um AS muito flexível que suporta vários métodos de autenticação.

Nas redes Wi-Fi aquando da autenticação de estações em que é necessário recorrer a certificados, existe a possibilidade do *supplicant* não dispôr dos certificados de raiz de *Certification Authority* (CA). Perante este facto, o que acontece na maioria dos casos é que em vez de proceder à sua instalação, o cliente opta por configurar o *supplicant* no sentido de que na fase da autenticação não proceda à sua verificação, ocorrendo assim uma autenticação bastante insegura que possibilita diversos tipos de ataques.

Focam-se aqui dois problemas que usualmente se encontram nas redes Wi-Fi, estando também a solução dividida em duas partes distintas, sendo que, as soluções complementam-se e visam a resolução de um problema comum que é a segurança de uma estação perante uma rede Wi-Fi forjada.

Para o primeiro problema, identificação do servidor de autenticação, a solução passa pela criação de uma nova semântica no SSID que se traduz na separação do mesmo em duas partes distintas, partes essas separadas por um caracter separador. Processo simples e engenhoso que resolve o problema.

Para o segundo problema identificado, o da certificação do servidor de autenticação, a solução é um pouco mais complexa passando pelo recurso ao ScalSec, novo modelo de certificação que recorre ao *Domain Name System Security Extensions* (DNSSEC).

1.2 Objectivos

O objectivo desta dissertação consiste na definição e implementação de uma solução de segurança que permita às estações 802.11 identificar correctamente redes Wi-Fi baseadas no servidor de autenticação, de forma a evitar redes forjadas sem que o utilizador tenha de introduzir mais informação para além do SSID, sendo que o AP não deve ser alterado e se possível, o servidor de autenticação também não.

1.3 Contribuições relevantes

As contribuições relevantes que esta dissertação introduz são as seguintes:

- **Definição de uma nova semântica para o SSID**, que permite identificar cada rede Wi-Fi (*ESS*) com um identificador único, permitindo a autenticação correcta do servidor de autenticação sem que para isso o utilizador tenha de introduzir informação adicional para além do SSID da rede desejada.

- **Definição de uma solução que recorre ao ScalSec**, proposta de uma *Public Key Infrastructure* (PKI) global que recorre ao DNSSEC, para certificar o servidor de autenticação sem alterar o mesmo, nem o AP. A solução permite certificar o servidor de autenticação a um custo financeiro reduzido, possivelmente nulo, com um esforço administrativo diminuto.

1.4 Estrutura da Dissertação

Esta dissertação está divida em oito Capítulos distintos, no decorrer dos quais tentar-se-à elucidar o leitor relativamente aos problemas e contratempos encontrados, bem como reportar-se-ão as soluções adoptadas.

- No primeiro Capítulo apresenta o enquadramento do problema identificado e os objectivos da dissertação.

- No segundo Capítulo é feita uma descrição mais detalhada dos problemas, focando-se também os objectivos e metas a serem atingidos.

- Por sua vez, no terceiro Capítulo será dado a conhecer o Estado da arte, que servirá como base para uma melhor percepção do trabalho. Sendo também efectuada uma abordagem aos trabalhos relacionados com as soluções propostas na presente dissertação.
- A descrição pormenorizada da arquitectura de segurança adoptada, ficou reservada para o quarto Capítulo.
- No quinto Capítulo, documenta-se o trabalho desenvolvido, constando aí também as decisões tomadas e alternativas rejeitadas. Procede-se também à explicação das ferramentas desenvolvidas e configuração das mesmas.
- Neste sexto Capítulo é realizada uma análise comparativa dos resultados das diferentes soluções.
- O sétimo e último Capítulo corresponde às conclusões, no qual se aborda o trabalho desenvolvido ao longo da dissertação, assim como as dificuldades sentidas e soluções encontradas. Aproveita-se ainda este capítulo para deixar sugestões de possíveis melhorias.
- Por fim os anexos, os quais contêm os programas que foram criados ou alterados, diferentes configurações para os servidores e scripts necessários para tornar o processo automático.

Capítulo 2

Definição do Problema

Num mundo de avanços tecnológicos em que as redes Wi-Fi estão em voga, torna-se pertinente esclarecer, embora de forma superficial, o funcionamento dessas mesmas redes.

Proceder-se-à de seguida à definição e caracterização de alguns termos que permitem ao leitor perceber o funcionamento das redes Wi-Fi, nomeadamente o *Basic Service Set* (BSS), *Service Set IDentifier* (SSID) e o *Extended Service Set* (ESS). Assim, o BSS corresponde à cobertura dada a uma ou várias estações que estão conectadas a um AP ou um grupo de APs e que partilham o mesmo identificador local. Por sua vez o SSID é o identificador local da rede Wi-Fi, estando normalmente associado a uma rede especifica. O ESS caracteriza-se por um ou vários APs que definem o mesmo SSID, ou seja, isto ocorre quando uma ou várias BSS estão interligadas, tendo o mesmo SSID.

Após análise e estudo da norma 802.11 um dos problemas identificados, que tem sido ignorado até hoje, é o facto de o SSID poder ser reutilizado por diferentes APs de diferentes ESS, sem no entanto a estação possuír forma de verificar se está, efectivamente, a utilizar a rede pretendida. O mesmo acontece quando a rede recorre a um servidor de autenticação (AS), dado que este não fornece um mecanismo que permita distinguir qual o ESS correcto. Este problema é grave pois o utilizador selecciona as redes a que se deseja associar apenas com base no SSID da rede, não tendo possibilidade de obter mais informações senão quando se associa perante a mesma. Após a associação à rede, a estação tentará sempre associar-se ao AP que apresente o melhor sinal podendo assim a estação mudar de rede (ESS) sem que o utilizador se aperceba.

Existem diversas topologias de redes Wi-Fi. As mais utilizadas em redes domésticas ou com poucos utilizadores tendem a recorrer a métodos que se baseiam em passwords, porque são conceptualmente mais simples e não apresentam qualquer custo financeiro. Por outro lado, as redes utilizadas em ambientes empresariais/académicos utilizam um servidor de autenticação para autenticar os diversos utilizadores. Os métodos de autenticação mais difundidos recorrem a um túnel *Transport Layer Security* (TLS) estabelecido entre a estação e o servidor de autenticação, o qual permite autenticar o servidor de autenticação através de um certificado e fornecer um canal seguro para autenticar o utilizador. Para que ocorra a autenticação na rede pretendida, de forma correcta, automática e sem intervenção do utilizador, o servidor de autenticação necessita de utilizar um

certificado de CA comercial, pois este já se encontra instalado na estação.

Embora os métodos de autenticação baseados no túnel TLS apenas requeiram um certificado de CA para o servidor, constata-se que muitas vezes os administradores, perante o custo financeiro e a complexidade do processo de certificação e instalação, tendem a recorrer a certificados *self-signed* para o servidor de autenticação. Por sua vez, os utilizadores em vez de instalarem localmente o certificado de CA no *supplicant*, tendem a desactivar a verificação dos certificados do servidor de autenticação. Esta falha na verificação permite a um atacante forjar uma ESS bastando para isso um AP com o SSID desejado e um servidor de autenticação. Desta forma as estações ao associarem-se ao AP forjado vão considerar o servidor de autenticação válido e estabelecer um túnel TLS, iniciando a autenticação do utilizador.

Visto que, tipicamente, os utilizadores não inserem o nome do servidor de autenticação esperado para cada rede Wi-Fi, a estação considera que o servidor com o qual a ligação foi estabelecida é o correcto, desde que este apresente um certificado emitido por uma CA de confiança. Um certificado de CA é considerado de confiança sempre que o seu certificado de raiz se encontrar instalado localmente no *supplicant*. Deste modo, normalmente, um servidor de autenticação é considerado como válido sempre que apresente um certificado emitido por uma CA comercial, sendo o único impedimento o preço. Neste sentido, um atacante necessita apenas de comprar um certificado a uma CA comercial, para o seu servidor de autenticação e configurar um AP com o SSID da rede que pretende forjar. A única vantagem em relação ao cenário anterior é o desincentivo que o atacante possui, pois necessita de gastar dinheiro para realizar o ataque.

Após ter derrubado a barreira de segurança mantida pelo túnel TLS, dependendo do método de autenticação utilizado para autenticar o utilizador, o atacante poderá tentar obter ou deduzir a password ficando assim com as credenciais do utilizador. Pode ainda indicar que a autenticação foi bem sucedida e depois observar e/ou alterar o tráfego do utilizador que passa pelo AP.

Face aos problemas identificados, pretende-se com a presente dissertação, definir e implementar uma solução que impeça um atacante de forjar uma rede Wi-Fi que utilize um servidor de autenticação, de modo a impedir que o atacante consiga obter ou deduzir a password do utilizador ou observe e/ou altere o tráfego do utilizador que passa pelo AP.

Assim e possuindo como tema de fundo a "Autenticação de redes Wi-Fi recorrendo ao DNSSEC", definiu-se como objectivo último o "Desenvolvimento de uma extensão para o *Wi-Fi Protected Access* (WPA) que permita o recurso ao DNSSEC para autenticação do servidor RADIUS e respectivas redes Wi-Fi".

Capítulo 3

Estado da Arte

Antes de se iniciar um trabalho com este perfil, é necessário um aprofundamento teórico dos conhecimentos, os quais permitem uma correcta compreensão das soluções adoptadas. Assim, é dado a conhecer de forma abrangente, neste Capítulo, o modo de funcionamento das normas e protocolos utilizados, desde a camada de rede até à camada de aplicação, sendo também realçadas as vantagens e fraquezas que se podem apontar aos mesmos.

Neste sentido, deu-se um enfoque aos protocolos de autenticação, aos certificados e seus processos de validação, ao *Domain Name System* (DNS)/DNSSEC, bem como à firewall da estação.

3.1 Criptografia

A crescente necessidade de segurança da informação transmitida, originou o nascimento da criptografia. Conceito que resultou da conjugação de duas palavras gregas "kryptós"e "gráphein", que significam respectivamente "oculto"e "escrever". Neste sentido, a criptografia engloba em si um conjunto de conceitos e técnicas que visam codificar informação, de modo a que, o acesso à mesma, seja somente permitido ao emissor e ao receptor, evitando assim, que um possível atacante consiga interpretá-la. As técnicas criptográficas são cada cada vez mais utilizadas, são um recurso comum, quando há necessidade de enviar informações importantes, para outro utilizador, através de uma rede pública. Assim e dentro deste contexto, a segurança da informação pode implicar:

- Privacidade/confidencialidade, integridade
- Anonimato ou identificação de identidades, origem das mensagens
- Certificação, revogação

Torna-se assim claro que um dos objectivos da criptografia passa por garantir que, numa troca de informação entre diferentes utilizadores, certos requisitos de segurança sejam cumpridos. Porém, para que isso fosse possível, sentiu-se necessidade de desenvolver protocolos, algoritmos, bem como legislação adequada.

3.1.1 Cifras

Uma cifra é um algoritmo criptográfico, que modifica o conteúdo da mensagem original de forma programática, obtendo-se assim um criptograma. No início da criptografia, os algoritmos das cifras eram secretos, sendo esse o factor crucial que definia grande parte da sua segurança. No entanto, na criptografia moderna, a segurança de uma cifra já não se consegue, apenas, tornando secreto o seu funcionamento, devido essencialmente, ao aumento de poder de processamento dos diversos sistemas de informação. Por outro lado, pode-se avaliar a qualidade de uma cifra pelo tempo que ela consegue permanecer inquebrável, sendo de conhecimento público, ou seja, logo que o algoritmo é conhecido, a sua robustez a ataques fica apenas dependente da chave e da sua qualidade. Assim é relevante explanar a directa ligação entre o tamanho da chave e a sua robustez, ou seja, se a cifra não possui fragilidades explicitas, quanto maior for o tamanho da chave, maior será o número de possibilidades o que dificultará a obtenção da mesma. Outro aspecto que contribui para dificultar ainda mais o processo, passa pela troca de chaves, com alguma regularidade.

3.1.2 Ataque a cifras

Num ataque a uma cifra, o adversário é colocado perante o desafio de descobrir a mensagem associada a um criptograma (mensagem codificada). Podemos assim, de um modo geral, caracterizar e classificar os ataques a cifras da seguinte forma:

criptograma conhecido – o atacante só conhece o criptograma enviado.

mensagem conhecida – adicionalmente, o atacante também obteve um determinado número de pares "mensagem/criptograma" (que não incluem o criptograma de desafio). Por vezes esse desafio é colocado de forma heurística : o adversário sabe que o criptograma resulta da cifra de uma de duas mensagens diferentes escolhidas por ele, só precisando assim, descobrir qual das duas foi cifrada.

escolha de mensagem – o atacante pode escolher quais as mensagens para as quais tem os respectivos criptogramas (tendo acesso à operação de cifra). É também uma escolha adaptativa quando é condicionada pelo desafio.

criptograma escolhido – o atacante pode escolher criptogramas para os quais pretende saber as mensagens associadas (desde que não seja o próprio desafio). Também aqui se distingue a versão adaptativa quando essa escolha depende do desafio.

Todas a cifras são vulneráveis a ataques de força bruta, os quais, consistem numa tentativa/erro de acertar na chave pretendida. Este tipo de ataques quase não são viáveis, isto porque,se o algoritmo utilizado tiver forte encriptação e número de bits suficiente, a obtenção da chave poderá levar várias dezenas de anos.

As cifras mais utilizadas operam em dois modos distintos, são eles:

Bloco – As mensagens são tratadas por unidades de blocos de dados, ou seja, é colocado um pré-determinado número de bits no buffer que é cifrado segundo regras definidas para cada cifra. Os tamanhos típicos para os blocos são: 64, 126, 256 bits, no qual a mensagem é partida em blocos do comprimento requerido. Devem, por isso, as mensagens ter um comprimento múltiplo do tamanho do bloco, ou convencionar-se que o último bloco é preenchido de acordo com uma regra pré-estabelecida (padding). A regra do padding depende de cada algoritmo, sendo que ela deve tornar claro quando o bloco de dados termina (tamanho do bloco de dados) e quando começa o padding.

Sequenciais – As mensagens são encriptadas recorrendo a uma chave pseudo-aleatória e de comprimento igual à mensagem. A chave pseudo-aleatória deverá obrigatoriamente ser sempre diferente, de comportamento imprevisível e nunca deve ser reutilizada ao longo do processo de encriptação. A chave pseudo-aleatória é gerada recorrendo a uma chave secreta fixa, que deverá produzir os mesmos resultados independentemente do hardware, assim são chaves finitas e é normal dizer-se que têm um período de tempo. O ataque a este modo de operação não é difícil, pois, mesmo dispondo só de alguns criptogramas pode retirar-se muita informação útil sobre a mensagem lá contida. Na teoria, o ideal é uma aproximação ao caso OTP (One Time Pad), que obtendo toda a mensagem de uma vez só encripta (executa um XOR entre a chave e a mensagem) com uma chave do mesmo tamanho uma única vez sem repetição. A verdade, é que muitas vezes não é possível ter uma chave do tamanho da mensagem e parte-se assim a mensagem em chaves de X bits, sendo atribuída a cada 'stream' de dados um dado estado da cifra. Neste sentido, pode-se partir as cifras sequenciais em dois tipos:

Síncronas – A sequência da chave é independente do criptograma e da mensagem, se forem retirados ou adicionados bits ao criptograma ele perde sincronismo e toda a mensagem a partir daquele ponto, ao ser decifrada será corrompida. Por outro lado, se alterarmos o criptograma alterando a ordem dos seus bits, a decifragem será bem sucedida, embora os bits na mensagem original estejam com a sua ordem trocada.

Auto-Síncronas – Neste caso, cada bit da nova chave é calculado com base nos bits anteriores do seu criptograma e chave. São adicionados delimitadores (bits aleatórios) na mensagem para permitir uma melhor sincronização, em caso de erro no criptograma apenas parte da mensagem se perde e a restante mensagem é sincronizada e decifrada.

No entanto, apesar das suas vantagens, este modo sofre de um problema, sendo vulnerável a ataques por repetição, no qual o atacante identifica os delimitadores e reenvia sucessivamente parte do criptograma.

No seguimento da análise destes dois tipos de processamento podemos concluir que, no processamento por blocos a unidade de processamento é distinta, mas por outro lado, são bem mais lentas que as cifras sequenciais. Por sua vez, o hardware necessário para as cifras sequenciais é menos complexo do que para as cifras por bloco, como consequência não protegem a chave,

devendo por isso, ser utilizada uma chave distinta para cada utilização. São também mais fáceis de atacar, uma vez que se utiliza uma chave com o mesmo número de bits para ambos os tipos de processamento. No que diz respeito aos ataques, nas cifras por blocos só podem ser considerados os ataques de "mensagem conhecida" e "escolha de mensagem", nas sequenciais todos os outros métodos podem ser considerados.

3.1.3 Cifras simétricas (Chaves simétricas)

As cifras simétricas constituem um sistema simples de encriptação de dados, neste sistema tanto o emissor como o receptor utilizam a mesma chave para a encriptação e desencriptação de dados. São conhecidos diversos tipos de chaves simétricas, no entanto, apenas serão enunciadas as principais: Twofish, Serpent, AES (também conhecido como Rijndael), Blowfish, CAST5, RC4, TDES (Triple DES) e o IDEA [19]. As cifras simétricas não necessitam de grande poder computacional, utilizam sempre uma chave secreta comum, a qual é normalmente mais fácil de obter do que a chave gerada. No entanto, um dos problemas destes algoritmos prende-se com a dificuldade de gestão das chaves secretas, sendo que, a troca de chaves secretas é um processo complexo e pouco flexível.

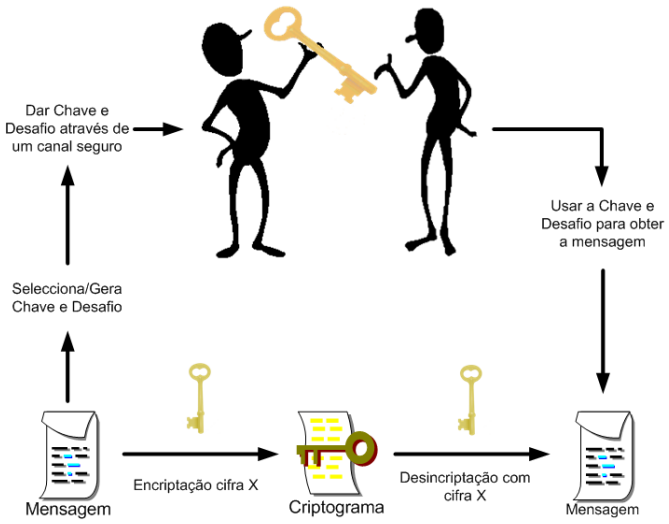


Figura 3.1: Distribuição de uma chave simétrica

3.1.4 Cifras assimétricas (Chaves assimétricas)

As cifras assimétricas também são conhecidas como sendo chaves públicas, utilizam funções matemáticas complexas de sentido único. Estas utilizam duas chaves, a chave pública e a chave privada. Ambas estão relacionadas matematicamente uma com a outra, no entanto, não é possível (quase impossível - pela complexidade da equação) obter a chave privada através da chave pública. Assim, é aplicado nas cifras assimétricas um sub-grupo destas funções matemáticas, as funções de sentido único com chave secreta *(trapdoor)*. Estas funções permitem, de forma eficaz, a inversão

da chave, através de uma chave secreta. O sistema de encriptação funciona da seguinte forma, o emissor, gera duas chaves uma chave pública e uma privada, associada a cada chave pública está um segredo (diferente para cada chave pública). Através da chave priElasvada, o emissor gera o criptograma e envia a chave pública com a chave secreta associada para o receptor, que por sua vez, aplica a função inversa à chave pública e decifra o conteúdo da mensagem. Sem conhecer a chave secreta, o receptor não consegue decifrar a mensagem. Alguns exemplos destas cifras assimétricas, são o caso da RSA, DSA, Diffie-Helman, Elliptic Curves. A utilização das cifras assimétricas está um pouco restrita, devido, essencialmente, ao poder computacional, necessário, para uma simples troca de mensagens, bem como devido à deficiência dos seus algoritmos. Por isso, a sua utilização é vista como um complemento as cifra simétricas, sendo, normalmente, executada uma negociação de chaves com cifras assimétricas (sessão) e a partir desse ponto, passa a utilizar-se a chave simétrica negociada. Note-se que se o objectivo for o de garantir a autenticidade então a figura 3.2 ilustra todo o processo necessário para se atingir esse objectivo, no entanto se o objectivo for o da confidencialidade, então nesse caso utiliza-se a chave pública para cifrar a mensagem e utiliza-se a chave privada do receptor para a decifrar.

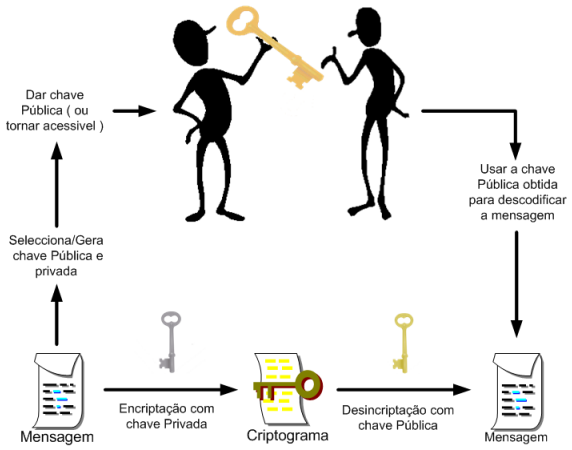


Figura 3.2: Distribuição de uma chave assimétrica

3.1.5 Hash criptográfico (Message digest)

No capítulo anterior, já se falou, das funções matemáticas de sentido único e no que consistem. No entanto, as cifras de hash criptográfico, ou message digest, utilizam essas funções para gerar uma chave fixa (hash), sendo ela única para uma dada mensagem de tamanho variável. Este tipo de cifras é essencialmente utilizado para garantir a integridade dos dados, podendo também, ser usadas para armazenamento de passwords e outros fins. Uma cifra de hash, deverá possuir as seguintes propriedades:

Pre-image Resistant – Deve ser bastante difícil(impossível),para um determinado valor de hash reverter a hash e obter a mensagem original.

Second Pre-image Resistant – Possuindo o valor de hash e a mensagem original, deverá ser muito difícil arranjar uma mensagem que consiga ter o mesmo valor de hash.

Collision Resitant – Deverá ser praticamente impossível que duas mensagens originem o mesmo valor de hash.

Esta última é uma das mais importantes em funções de hash criptográfico. Relativamente à sua segurança, a análise probabilística estabeleceu um número mínimo de 128 bits para que as cifras sejam resistentes a ataques, sendo que, modos mais sofisticados de ataques foram descobertos, sendo actualmente recomendadas cifras com mais bits, nomeadamente 160. As cifras mais utilizadas baseiam-se em cifras por blocos e usam uma construção genérica do tipo Merkle-Damgård a qual pode ser constatado na figura 3.3. Os tipos de cifra de hash criptográfico mais conhecidos, são o md4, md5, SHA-I / II, MAC, HMAC.

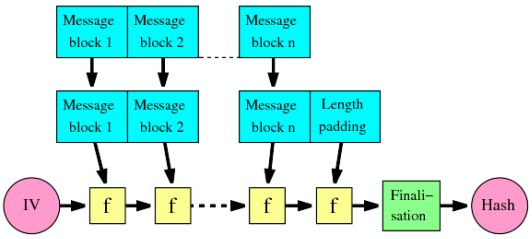


Figura 3.3: Esquema de blocos de Merkle-Damgård

3.1.6 Tamanhos de Chaves

No decorrer deste capítulo, abordou-se a temática de criptografia e suas cifras, sendo o tamanho da chave um dos pontos focado, quer em cifras simétricas, quer em cifras assimétricas, como parte importante na segurança das mesmas. Devido a sua importância, o tamanho das chaves é estudado por diversas organizações, as quais produzem regularmente um manual de boas práticas. Esse manual, tem em conta o tipo de cifra, o tempo de utilização e o fim a que se destina, sendo com base nesses parâmetros que é indicado um tamanho de chave mínimo. Assim, o tamanho da chave deve ser baseado nos manuais existentes pois eles são sem dúvida uma referência importante na definição do tamanho a adoptar.

3.2 Certificados

Esforços recentes para melhorar a segurança na Internet levaram ao aparecimento de um grupo de protocolos (S/MIME, IPSec, etc.) que utilizam a criptografia de chave pública para garantir:

- Confidencialidade
- Integridade

- Autenticação
- Não repúdio.

Esta utilização baseia-se no conceito de Certificado *(de chave pública)*.

A PKI tem como objectivo a gestão de chaves e certificados de forma segura e eficiente, de modo a permitir essa mesma utilização. A recomendação X.509 da *International Telecommunications Union* (ITU) é que orienta a utilização da criptografia de chave pública nas telecomunicações.

A aplicação dessa recomendação à Internet está definida num conjunto de *Request for Commentss* (RFCs) publicados pela *Internet Engineering Task Force* (IETF). Dentro da IETF, o grupo que gere os RFCs relacionados com o X.509 chama-se "PKIX Working Group". Este grupo de trabalho mantém um conjunto de documentos que se denomina "Internet X.509 Public Key Infrastructure".

Os utilizadores de um sistema baseado numa PKI devem poder confiar que, cada vez que utilizam uma chave pública, o agente com quem querem comunicar possui a chave privada associada. Esta confiança constrói-se com base em certificados de chave Pública, os quais correspondem a uma estrutura de dados que associa uma chave pública a um determinado agente (a uma representação da sua identidade).

A associação chave/agente é estabelecida por uma entidade terceira, uma autoridade de certificação que assina digitalmente cada certificado. A utilidade de um certificado depende unicamente da confiança depositada na autoridade de certificação.

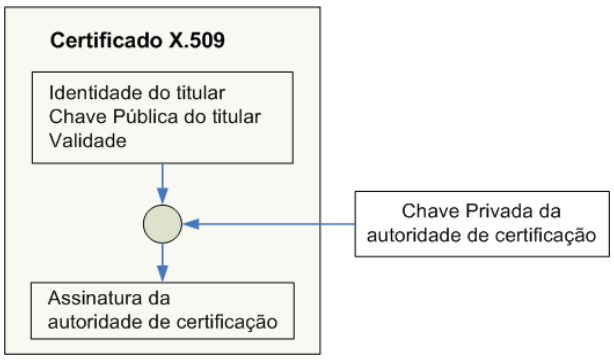


Figura 3.4: Esquema de combinação entre CA e Titular

O utilizador do certificado confia que a Autoridade de certificação verificou que a chave pública contida no certificado pertence de facto ao titular, sendo que, a assinatura da CA assegura a autenticidade e integridade do mesmo.

Um certificado de chave Pública é válido durante um período de tempo bem definido, o qual vem especificado no conteúdo assinado.

Como a assinatura e a validade temporal de um certificado podem ser verificados independentemente, por um utilizador, os certificados podem ser distribuídos por canais inseguros.

Os certificados de chave pública são utilizados maioritariamente, na validação de informação assinada digitalmente. Este processo consiste, geralmente, nos seguintes passos efectuados pelo destinatário:

1. verificar que a identidade indicada pelo emissor está de acordo com a identidade indicada no certificado.
2. verificar que o certificado é válido:
   - que a assinatura do certificado é válida;
   - que foi efectuada por uma autoridade de certificação de confiança;
   - que o certificado está dentro do seu período de validade.
3. verificar que a informação que recebe está de acordo com as permissões/privilégios do emissor.
4. utilizar a chave pública contida no certificado para verificar a assinatura da informação recebida.

Se todos os passos, acima referidos, forem executados sem problemas, o destinatário aceita que a informação foi assinada pelo emissor e que permanece inalterada.

Para protecção de informação ao nível da confidencialidade o certificado passa a ser utilizado pelo emissor contendo a informação relativa ao destinatário, sendo que o emissor executa os seguintes passos:

1. valida o certificado e a identidade do destinatário;
2. utiliza a chave pública contida no certificado para cifrar a informação;
3. envia a informação cifrada ao destinatário que a decifra com a sua chave privada.

3.2.1 Certificados X.509

No seguimento do que tem vindo a ser abordado, proceder-se-á a um aprofundamento histórico e técnico sobre diferentes versões dos certificados X.509. Sendo que, foi em 1988 que surgiu a primeira versão, contendo os seguintes campos:

- Version
- Certificate serialNumber
- AlgorithmIdentifier signature
- Name issuer
- Validity
- Name subject

– SubjectPublicKeyInfo

A assinatura destes certificado era efectuada pela Autoridade de Certificação *(CA)* sobre uma codificação DER.

Por sua vez, a segunda versão surge em 1999 não chegando a ser muito utilizada, devido, essencialmente, ao surgimento, pouco depois, da versão actual (V3). Na segunda versão foram introduzidos dois novos campos:

– UniqueIdentifier issuerUniqueID

– UniqueIdentifier subjectUniqueID

Estes campos tentavam rectificar a dificuldade existente em garantir que os campos do tipo Name tivessem valores únicos. Neste sentido, a IETF recomendou que as CAs não utilizassem estes campos e que garantissem, na medida do possível, a unicidade dos nomes. Por outro lado, recomendou também que, caso existam, estes campos não devem ser ignorados.

Nos dias de hoje utiliza-se a terceira versão, a qual foi normalizada em 1996 e é compatível com as versões anteriores. Esta versão, veio colmatar deficiências identificadas nas anteriores, relativamente a algumas aplicações, as quais passavam pela necessidade de mais atributos.

Como novidade, esta versão introduziu um novo campo do tipo extensões, basicamente, uma colecção de elementos do tipo extensão. As quais permitem a associação de atributos genéricos a um agente ou à sua chave pública, de forma muito flexível, sendo ainda, que cada extensão é ela própria uma estrutura de dados com um identificador e um valor adequado ao tipo do atributo que representa.

Enumeram-se de seguida os atributos desta terceira versão:

- **version**, tem de estar de acordo com o conteúdo do certificado.
- **serialNumber**, número único atribuído pela CA.
- **signature**, estrutura que identifica o algoritmo utilizado para gerar a assinatura da CA que acompanha o certificado.
- **validity**, estrutura com as duas datas que delimitam o período de validade do certificado.
- **subjectPublicKeyInfo**, estrutura contendo a chave pública do titular do certificado e identificação do algoritmo correspondente.
- **issuer e subject**, identificam respectivamente, a CA e o titular do certificado. Ambos são do tipo Name, o qual provém da norma X.501 e é ainda utilizado porque permite a compatibilidade com os sistemas de directório definidos nas normas X.500 (e.g. DAP e LDAP).

O tipo *Name* é uma colecção de atributos, geralmente strings da forma "< nome > = < valor >". Um conjunto destes atributos define um Distinguished Name para o agente titular. O Distinguished

Name tem uma estrutura hierárquica, inclui por exemplo, o país da organização e o nome próprio do agente ou entidade.

A norma X.520 dita alguns dos componentes de um Distinguished Name. Algumas aplicações importantes utilizam também o endereço de email como um dos atributos centrais da construção do Distinguished Name, mas os seguintes componentes são os que são de reconhecimento obrigatório, sendo os mais utilizados:

- country (C)
- organization (O)
- organizational-unit (OU)
- common name (CN)
- serial number (SN)

As extensões definidas na terceira versão do X.509 podem ser marcadas como Critical ou Non Critical. O uso dessa extensão permite, por exemplo, que quando uma aplicação encontre uma extensão crítica que não esteja identificada no certificado seja obrigada a rejeita-la, não sendo também permitidas várias instâncias da mesma extensão.

O IETF normaliza as extensões recomendadas para utilização na Internet em [10], definindo o identificador (OBJECT IDENTIFIER) e o tipo de dados associados. São desaconselhados desvios desta recomendação, nomeadamente, no que diz respeito a extensões críticas, apesar de não haver qualquer limitação a nível do standard.

De seguida, serão abordadas um conjunto de extensões suportados pelo X.509, como seja:

Subject Key Identifier , serve para identificar o certificado que contém uma determinada chave pública ex. quando um agente tem várias. É, em geral, um valor de hash derivado da chave pública que, caso esta extensão não conste do certificado, pode ser calculado em run-time.

Authority Key Identifier , serve para identificar a chave pública da CA que assinou o certificado, caso existam várias, o que facilita a verificação de cadeias de certificação. Isto pode ser feito:

- identificando o certificado da CA através do seu Subject e Serial Number;
- ou através da extensão Subject Key Identifier do certificado da CA.

Subject/Issuer Alternative Name , permitem associar formas de identificação alternativas ao titular do certificado ou à CA que o emitiu, e.g. email, nome DNS, endereço IP, URI, etc.

Basic Constraints , permite assinalar um certificado como pertencendo a uma CA e limitar o comprimento de cadeias de certificados.

Key Usage , esta extensão permite restringir as utilizações do par de chaves associado ao certificado e.g., quando uma chave apenas pode ser utilizada para verificar assinaturas digitais. Esta extensão contempla as seguintes utilizações:

- ***digitalSignature*** - Assinaturas digitais para autenticação e protecção da integridade de dados, excepto certificados e CRLs.
- ***nonRepudiation*** - Assinaturas digitais para não repúdio.
- ***keyEncipherment*** - Protecção da confidencialidade de chaves.
- ***dataEncipherment*** - Protecção da confidencialidade de dados.
- ***keyAgreement*** - Protocolos de acordo de chaves.
- ***keyCertSign*** - Assinatura de certificados.
- ***cRLSign*** - Assinatura de CRLs.
- ***encipherOnly/decipherOnly*** - Restringem a funcionalidade keyAgreement

Extended Key Usage , permite especificar ou restringir as utilizações previstas para o par de chaves associado ao certificado, em adição ou em alternativa à extensão Key Usage. Estão definidas diversas utilizações, bem como, a sua relação com as especificadas na extensão Key Usage:

- WWW server authentication
- WWW client authentication
- Signing of downloadable executable code
- E-mail protection
- ...

CRL Distribution Points , serve para indicar ao utilizador de um certificado onde pode obter informação quanto à revogação do mesmo, na forma de Certificate Revocation Lists (CRLs).

Embora existam outras extensões, não serão aqui abordadas pois não são pertinentes para o decorrer e compreensão do trabalho.

3.2.2 PKI

A Public Key Infrastructure define-se como um conjunto de hardware, software, pessoas, políticas e procedimentos necessários para criar, gerir, armazenar, distribuir e revogar Certificados de chave pública.

São cinco os componentes que constituem uma PKI, nomeadamente:

- Titulares de Certificados de chave pública que possuam chaves privadas, utilizando-as para decifrar mensagens e assinar documentos.

- Clientes que utilizam a chave pública contida num certificado, para cifrar mensagens e verificar assinaturas.
- Autoridades de Certificação que emitem e revogam certificados.
- Autoridades de Registo que garantem a associação entre chaves públicas e identidades de titulares (são opcionais).
- Repositórios que armazenam e disponibilizam certificados e CRLs.

Relativamente ao funcionamento de uma PKI, este, baseia-se em dois tipos de protocolos:

- Protocolos Operacionais, necessários para entregar certificados e CRLs aos sistemas que os utilizam, podendo estas operações ser efectuadas de diversas formas, incluindo o LDAP, HTTP e FTP. Para todos estes meios estão especificados protocolos operacionais que definem, inclusivamente, os formatos das mensagens.
- Protocolos de Gestão, necessários para dar suporte às interacções entre os utilizadores e as entidades de gestão da PKI, nomeadamente:
  - Inicialização.
  - Registo e Certificação.
  - Recuperação e Actualização de pares de chaves.
  - Pedido de revogação.
  - Certificação de CAs.

3.2.3 Cadeias de Certificação e Confiança

Para utilizar um serviço que requer o conhecimento de uma chave pública, é necessário obter e validar um certificado que a contenha. Por sua vez, a validação do certificado implica o conhecimento da chave pública da Autoridade de Certificação *(CA)* que o emitiu e, consequentemente, a obtenção e validação do certificado que a contém.

A validação do certificado raiz de CA, poderá implicar o conhecimento da chave pública de outra CA que o tenha emitido e assim sucessivamente. Chama-se a esta sequência uma Cadeia de Certificação. Em geral, para se conhecer e confiar numa chave pública é necessário validar o certificado dessa chave.

Cada utilizador, conhece um número limitado de chaves públicas pertencentes a CAs (em geral Root CAs), as quais funcionam como raízes das relações de confiança. Isso significa que, o utilizador aceitará um certificado emitido por uma dessas CAs e que depositará um determinado nível de confiança no seu conteúdo. Assim, a validação de uma cadeia de certificados terminará quando for encontrado um certificado com essa característica. O grau de confiança depositado num certificado validado, baseia-se apenas, na confiança depositada na CA que funcionou como

raiz da relação de confiança. As Certificate Revocation Lists (CRL) são o canal previsto na norma X.509 para revogação de certificados dentro do período de validade.

Uma CRL pode designar-se por:

- Base CRL - quando lista todos os certificados revogados por uma CA, que ainda estão no seu período de validade.
- Delta CRL - quando apenas lista os certificados revogados, a partir da publicação de uma Base CRL referenciada.

As CRLs são emitidas, em geral, pelas próprias CAs. No entanto, é possível que a CA delegue esta função numa outra autoridade denominada CRL Issuer. Uma CRL emitida nestas condições denomina-se CRL Indirecta. Cada CRL tem um contexto específico, ou seja, um conjunto de certificados passíveis de aparecerem no seu conteúdo. Assim, para que uma CRL possa ser utilizada eficientemente, este contexto deve estar bem definido.

Relativamente à sua constituição, uma CRL é composta pelos seguintes campos:

- Version
- AlgorithmIdentifier signature
- Name issuer
- Time thisUpdate
- Time nextUpdate
- Extension fields

A assinatura da CRL é efectuada pela CA, sobre uma codificação DER da representação desta estrutura de dados em ASN.1.

O campo *version* é opcional e deve ter o valor 2 uma vez que as CRLs foram introduzidas na versão 2 do X.509. O campo *signature* tem o mesmo significado que nos certificados. A segurança de uma PKI depende, da eficácia com que são revogados os certificados que se tornaram inválidos. Este facto sugere que, assim que um certificado se torna inválido uma nova CRL deve ser publicada.

3.2.4 OCSP

O protocolo *Online Certificate Status Protocol* (OCSP) especificado em [17], é sem dúvida, um passo em frente na verificação da revogação dos certificados. É uma alternativa viável, que permite obter informação sobre o estado de revogação de um determinado certificado. Possui informação recente, introduzindo menos encargos à rede e à aplicação no processamento dessa informação, obtendo ainda, funcionalidades adicionais, em comparação com o uso de CRLs.

Devido ao modo de operação do protocolo, do tipo "request/response" o servidor OCSP é normalmente chamado de *responder*. Desta forma, o cliente OCSP ao emitir um pedido ao *responder*

OCSP, suspende, temporariamente, a aceitação do certificado até à obtenção de uma resposta. A resposta do *responder* é assinada, retornando os seguintes resultados: “good”, “revoke”, “unknown”. Podendo, além disso, retornar ainda códigos de erro, tornando a identificação do erro mais fácil.

Por seu lado, os pedidos OCSP são bastante versáteis, podendo ser configurados para determinados esquemas de PKIs através do uso de extensões adicionais. Para uma verificação OCSP ser possível, é necessário que a CA tenha a extensão authorityInfoAccess preenchida, sendo essa extensão que indica ao cliente qual o método de acesso, especificando o tipo e o formato da informação do Issuer através do campo accessMethod, bem como, qual o endereço a aceder através do campo accessLocation, este último pode ser do tipo HTTP, LDAP, FTP, etc.

Os pedidos OCSP contêm os seguintes campos:

- Versão do protocolo
- Identificador do certificado
- Identificação do serviço requisitado
- Extensões opcionais

Sendo assim, o pedido OCSP é constituído por uma lista de estruturas ASN.1 CertID, que contêm os seguintes campos: hashAlgorithm, issuerNameHash, issuerKeyHash, serialNumber. O Issuer do certificado de raiz de CA é identificado através do valor da hash do Distinguished Name e da sua chave pública.

No *Responder*, o pedido é verificado seguindo as seguintes etapas:

- Verifica o formato e sintaxe da mensagem.
- Verifica que o servidor está configurado para fornecer os serviços pretendidos.
- Verifica se o pedido tem toda a informação necessária para processar
- Construção da resposta.

Relativamente às respostas aos pedidos OCSP, estas são formadas pelos seguintes campos:

- A versão sintaxe da resposta
- O nome do *responder* OCSP
- Lista de respostas para cada um dos certificados especificados na lista de pedidos, sendo que, para cada um desses certificados será retornada a seguinte informação:
  - O identificador do certificado.
  - O estado do certificado, sendo eles: good, revoked ou unknown.
  - Período de validade da resposta.

– Uma assinatura digital da resposta, a qual poderá ser emitida por uma das seguintes entidades:

  - O *Issuer* do certificado de raiz de CA que emitiu o certificado em questão.
  - Um Trusted Responder, por exemplo, um *responder* em cuja chave pública o cliente confie.
  - Um CA Designated Responder, por exemplo, um *responder* autorizado pelo *Issuer*, do certificado de raiz de CA que emitiu o certificado. Neste caso específico é necessário que o certificado de raiz CA tenha a extensão extendedKeyUsage com campo id-kp-OCSPSigning activado.

Torna-se necessário clarificar que, o significado da resposta good, apenas, se destina a indicar que não existe qualquer tipo de revogação daquele certificado, podendo o mesmo até nem existir, no entanto, a resposta seria a mesma. No que se refere ao período de validade de uma resposta, este, pode ser efectuado recorrendo a três indicações:

thisUpdate – Indica a hora a que o responder sabe que a resposta é correcta.

NextUpdate – Indica a hora a que o responder sabe que poderá fornecer uma resposta actualizada. Se tal não for indicado, indica que qualquer nova resposta contém informação actualizada.

ProducedAt – Indica a hora a que o responder assinou a resposta.

O cliente OCSP, para validar uma resposta OCSP necessita, obrigatoriamente, de verificar que:

– Os certificados do pedido sejam os mesmos indicados na resposta.

– A assinatura da resposta é válida e provém de um agente autorizado.

– A resposta é suficientemente recente, não ultrapassando a sua validade.

Esta versão do protocolo é normalmente conhecida como sendo a versão um. De seguida, iremos explicar melhor em que consiste a nova versão ou também conhecida por segunda versão.

Nesta segunda versão uma das alterações mais significativas, passa pelo comportamento de verificação do OCSP perante uma ligação segura (TLS), podendo esta, ser do tipo explícito ou implícito. Assim, se um cliente TLS requerer uma verificação do estado do certificado, ele deve, adicionar na mensagem de hello o texto status_request, se tal não acontecer, o cliente, pode ainda proceder à verificação do certificado durante a negociação do handshake, resultado este, que virá no campo CertificateStatus. O primeiro caso, é normalmente designado de *OCSP Stapling*, no qual o servidor de TLS vai requerendo ao OCSP responder o estado do seu certificado de CA e quando tem uma ligação TLS envia a resposta OCSP juntamente com o seu certificado. O segundo caso é mais ineficiente, ou seja, o cliente TLS efectua um pedido OCSP, pedido executado pelo servidor TLS, que posteriormente lhe passa a resposta, facto que pode provocar uma sobrecarga no servidor de TLS.

Esta segunda versão publicada pelo IETF, permite adicionar funcionalidades avançadas que podem ser sucintamente descritas, na definição de três novos serviços, os quais são indicados nos pedidos OCSP através do uso do campo requestExtensions. Os seguintes serviços foram assim definidos:

Online Revocation Status (ORS) – Este serviço ocorre por defeito, ou seja, pressupõe que nenhum outro serviço é requerido, fornecendo assim, a funcionalidade básica acima descrita.

Delegated Path Validation (DPV) – Com a adição deste serviço, o cliente OCSP delega para um outro servidor o processamento e a validação das cadeias de certificados. Desta forma, o cliente OCSP pode assim, ter apenas um número mínimo de funcionalidades para a verificação dos certificados, sem que, no entanto, perca o controle sob a forma como o servidor irá proceder à validação dos mesmos, o que significa que pode continuar a fornecer políticas de certificação ou restrição de cadeias, entre outras.

Delegated Path Discovery (DPD) – Com este serviço o cliente OCSP para além de validar os certificados pode obter do servidor diversas informações sobre os mesmos. As informações sobre a revogação desses certificados pode ser feita recorrendo a servidores exteriores usando LDAP, X.500, FTP, HTTP, através da utilização OCSP ou CRL's. Sendo possível, também, definir regras e políticas de certificação para cada cadeia de validação.

3.2.5 PKCS

A empresa RSA Data Security formada pelos inventores das técnicas RSA de criptografia de chave pública, tem um papel importante na criptografia moderna e especialmente a sua divisão RSA Laboratories que mantém uma série de standards denominada Public Key Cryptography Standards (PKCS) muito importantes na implementação e utilização de PKIs.

Os PKCS visam preencher o vazio que existe nas normas internacionais, nomeadamente, a nível de formatos para transferência de dados que permitem a compatibilidade/interoperabilidade entre aplicações que utilizam criptografia de chave pública. São doze os standards deste tipo: PKCS#1, #3, #5, #6, #7, #8, #9, #10, #11, #12, #13 e #15.

Com a publicação dos standards, a RSA tem como objectivos, segundo os próprios:

- Manter a compatibilidade com os standards existentes, nomeadamente com PEM (Privacy-Enhanced Mail Protocol).
- Ir além dos standards existentes permitindo, entre aplicações, melhor integração e mais completa. Normalizando a troca segura de qualquer tipo de dados.
- Produzir um standard que possa ser incluído numa futura versão dos standards OSI (Open Systems Interconnection).

Os PKCS, utilizando o ASN.1, descrevem a sintaxe de mensagens de uma forma abstracta, não restringindo a sua codificação. Essa descrição passa por, especificar os algoritmos criptográficos e a sintaxe de mensagens de forma ortogonal.

3.3 Extensible Authentication Protocol (EAP)

O protocolo *Extensible Authentication Protocol* (EAP) foi definido em [2], como sendo um protocolo bastante flexível e que permite vários métodos de autenticação, não podendo, no entanto, ser considerado um mecanismo de autenticação específico, mas sim, um *framework* universal de autenticação. Concebido inicialmente para *Point-to-Point Protocol* (PPP), tem agora frequente utilização em redes Wi-Fi, tendo sido declarados cinco métodos de autenticação oficiais para autenticação de redes "Enterprise" de WPA e *Wi-Fi Protected Access 2* (WPA2).

O EAP não é um protocolo rígido, define antes, o formato das mensagens trocadas entre as estações e os servidores de autenticação (*Autentication Server* (AS)). Cabe aos protocolos que utilizam o EAP definir o modo de encapsulamento das mensagens EAP, por exemplo, no caso de 802.1X, o encapsulamento é chamado de EAPOL (EAP Over LANs). Actualmente, com o EAP podem ser utilizados vários métodos de autenticação, no entanto, são quatro aqueles que se destacam:

***EAP-MD5* –** Baseado em passwords

***EAP-TLS (Transport Layer Security)* –** Baseado em certificados (Cliente/Servidor)

***EAP-TTLS (Tunneled TLS)* –** Certificados apenas no Servidor

***EAP-PEAP (Protected EAP)* –** Permite dois tipos de autenticação distintos (excluindo drafts):

- Microsoft CHAPV2 (Certificados apenas no Servidor)
- TLS (Certificados Cliente / Servidor)

	EAP MD5	EAP TLS	PEAP	EAP TTLS
Autenticação Mútua	Não	Sim	Sim	Sim
Certificados no Servidor	Não	Sim	Sim	Sim
Certificados no Cliente	Não	Sim	Não	Não
Segurança	Fraca	Super Forte	Forte	Forte
Requer PKI	Não	Sim	Não	Não
Dynamic Key Exchange	Não	Sim	Sim	Sim

Tabela 3.1: Analise dos métodos de autenticação EAP

O protocolo EAP foi adaptado para 802.1X sendo encapsulado pelo 802.1X. É o protocolo de autenticação mais utilizado em redes 802, incluindo as redes Wi-Fi. A adaptação foi feita de forma a que, as alterações necessárias aos intervenientes utilizados numa comunicação Wi-Fi, fosse mínima. Por exemplo, no caso do AP, ele não se encontra directamente envolvido no processo de autenticação, apenas permite a passagem de tráfego EAP. Uma das vantagens subjacentes é que a utilização de diferentes protocolos de autenticação, não implica nenhuma alteração nos APs. Convém ainda referir que, o EAP pode também ser utilizado para distribuição dinâmica de chaves, como sejam, o caso das chaves *Wired Equivalent Privacy* (WEP)/WPA.

No sentido de clarificar o processo EAP será exposto na figura 3.5 a troca de mensagens que é feita numa autenticação EAP em redes Wi-Fi. A estação utilizará um *supplicant* para se conectar ao AP, assim é definido pela 802.1X. O *supplicant* terá de ser configurado de acordo com as configurações do servidor de autenticação o qual pode ser configurado de maneira a apenas permitir determinados tipos de enlace naquela estação.

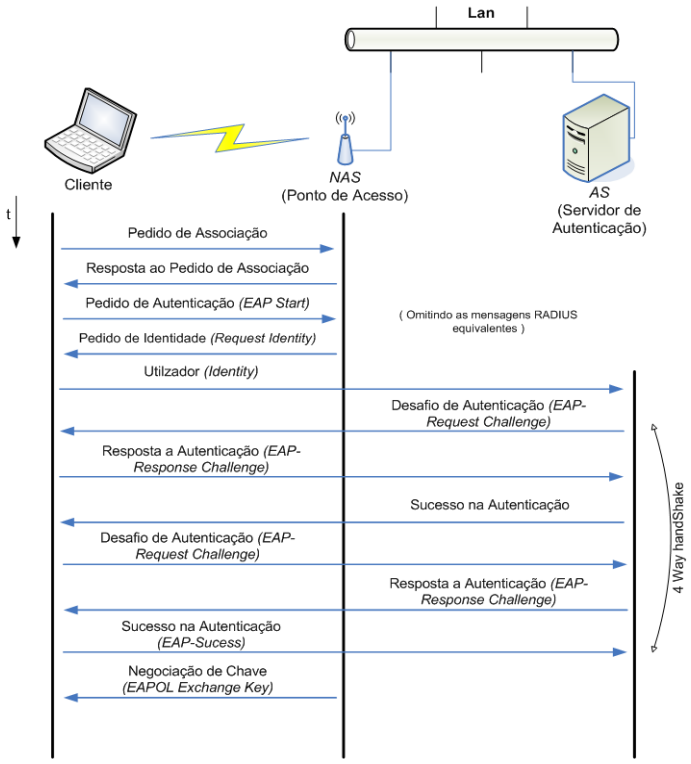


Figura 3.5: Autenticação EAP

Porém, nem só de vantagens se fez a adaptação do protocolo às redes Wi-Fi, ou seja, este não foi concebido para redes sem fios, mas sim, pensado para redes cabladas. Como tal, não tem qualquer noção de segurança ou confidencialidade em redes 'abertas', devendo, por isso, durante a fase de autenticação, a comunicação entre as estações e os APs estar protegida com chaves de cifra, nomeadamente WEP, ou *Temporal Key Integrity Protocol* (TKIP). Outro dos problemas identificados, surge quando não ocorre uma autenticação mútua entre a estação e o AP, podendo uma estação ser enganada por um AP mais potente.

3.4 RADIUS

O RADIUS é um protocolo cliente/servidor denominado de *Authentication, Authorization, Accounting* (AAA), que devido ao seu modo de implementação, pode funcionar em ambas as aplicações, locais ou de mobilidade. Normalmente, este protocolo é utilizado por aplicações que fornecem acesso à rede para serviços do tipo:

- Dial-up

- Rede privada virtual (VPN)

- Redes sem fios (Wi-Fi)

O RADIUS dispõe de diversas alternativas de autenticação, ou seja, o utilizador pode fornecer as suas credenciais através de um desafio/resposta ou pode ainda utilizar protocolos PAP, CHAP ou mesmo EAP. O servidor RADIUS obtém os dados e processa a informação fornecida pelo utilizador, verificando se estes estão correctos, bem como, a que serviços está o utilizador autorizado a aceder.

Apesar de não ter sido desenvolvido, especificamente, para servir como método de autenticação para redes Wi-Fi, oferece um melhoramento sobre o WEP padrão em conjunção com outros métodos de segurança como o EAP-PEAP, ou EAP-TTLS, entre outros.

Um exemplo do modo de operação do protocolo RADIUS, consiste na necessidade de introdução de credenciais, por forma a que lhe seja autorizado o acesso, sempre que um utilizador estabelece uma ligação com um *Internet Service Provider* (ISP), podendo ser por modem, DSL, cabo ou Wi-Fi. Essas credenciais são normalmente o username e password, podendo também incluir a negociação de certificados por parte do cliente. Esses dados são posteriormente encaminhados para um servidor de acesso à rede (Network Access Server - NAS) e depois para o servidor RADIUS (AS), utilizando o protocolo RADIUS. Os dados trocados entre o NAS e o servidor RADIUS estão cifrados usando uma chave pré-combinada e um algoritmo de hash MD5. O servidor RADIUS verifica a autenticidade destas credenciais através de protocolos de autenticação, PAP, CHAP ou EAP. Se as credenciais forem aceites o RADIUS autorizará o acesso à rede. Focando-se o caso das redes sem fios, o cliente do servidor de autenticação RADIUS será o AP que funciona como um NAS, enviando pedidos de autenticação e registo ao servidor RADIUS. Outros exemplos do uso do RADIUS ocorrem no VoIP, onde é utilizado durante a passagem de credenciais, ou no RSA SecurID.

Ao contrário do EAP, o RADIUS já precisa de uma rede IP especificada para funcionar. Desta forma, o IETF decidiu atribuir os portos 1812 e 1813, para a autenticação e gestão (accounting), respectivamente. O protocolo define também uma norma de sinalização que permite aos dispositivos aceder às funcionalidades comuns dos diferentes servidores de autenticação. O protocolo RADIUS encontra-se definido em [20] e [21].

Há portanto que distinguir entre um servidor RADIUS, que é um servidor de autenticação que suporta as capacidades RADIUS e o protocolo RADIUS que é usado para comunicar com o servidor.

Actualmente já existe uma alternativa ao RADIUS, é o protocolo DIAMETER que tem como objectivo substituir o RADIUS, sendo que, uma das diferenças entre os dois, é que, enquanto que o RADIUS utiliza UDP, o DIAMETER utiliza *Secure Transmission Control Protocol* (STCP), ou o *Transmission Control Protocol* (TCP).

3.5 TLS/SSL

O *Secure Socket Layer* (SSL) está para o TCP como o Ipsec está para o IP. É uma melhoria da camada de transporte para incluir segurança nas comunicações. O *Secure Socket Layer* (SSL) foi desenvolvido pela Netscape e a sua versão 3 foi adoptada em 1999 pela IETF sob a designação *Transport Layer Security* (TLS). O TLS foi definido em [8], designando-se por TLS v1.0 que apesar de baseado no SSL v3.0, não é totalmente compatível, por exemplo, descontinuou o suporte do Fortezza cipher suite e códigos de alertas adicionais. Em 2006 saiu uma nova revisão do TLS, o TLS v1.1, a qual é definida em [9] e vem resolver alguns problemas apontados ao TLS V1.0.

Relativamente aos serviços fornecidos pelo TLS, estes passam por:

- Confidencialidade, baseando-se em cifras simétricas.
- Autenticação, utilizando para isso criptografia de chave pública.
- Integridade, utilizando para esse efeito o Message Authentication Codes (MAC).

No que se refere ao SSL, este encontra-se estruturado em duas sub-camadas:

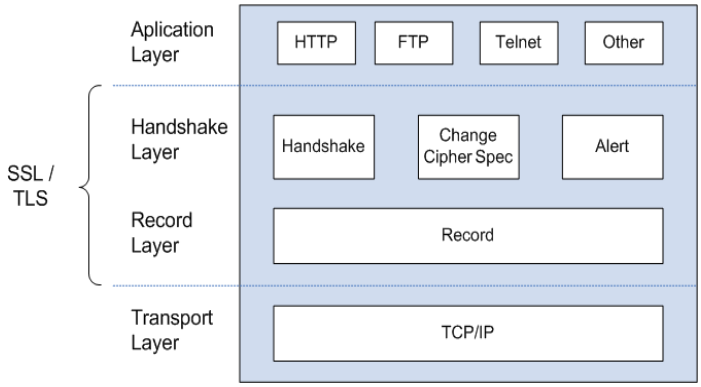


Figura 3.6: TLS na camada OSI

Handshake Layer , permite a autenticação mútua entre clientes e servidores antes de se iniciar a troca de dados através da Record Layer.

Record Layer , encapsula a informação correspondente às camadas superiores.

Na camada *Handshake Layer*, podemos identificar 3 sub-protocolos:

Handshake - Usado para negociar a sessão entre os dois lados da comunicação, nomeadamente a negociação de algoritmos e chaves criptográficos, assim como, a troca dos certificados digitais, caso seja necessário.

Change Cipher Spec - Mensagem para notificar o receptor que as próximas mensagens enviadas serão encriptadas com as chaves recém negociadas.

Alert - Mensagens trocadas para informar um erro (ex: mensagem não pode ser obtida) ou simplesmente informativas (ex: negociação rejeitada)

O funcionamento do SSL baseia-se em sessões que são estabelecidas entre um cliente e um servidor. Cada sessão SSL pode incluir várias ligações seguras e cada cliente pode manter diversas sessões SSL. Durante o seu estabelecimento e operação, as sessões e ligações SSL atravessam uma sequência de estados. Ou seja, existe uma máquina de estados, quer no cliente, quer no servidor, para cada sessão e ligação, sendo a camada de Hanshake a responsável pela sincronização dos estados no cliente e no servidor.

Relativamente as transições entre estados, estas efectuam-se em duas fases:

- Primeiro - constrói-se/negoceia-se um pending state.
- Depois - substitui-se o operating state pelo pending state.

Em concordância com o que foi referido anteriormente e por forma a permitir uma melhor compreensão do estado de uma ligação enumeram-se de seguida, alguns nomes técnicos que ocorrem durante a mesma: TLS.

Compression method - Algoritmo de compressão da informação antes de ser cifrada.

Cipher spec - Algoritmo de cifra simétrica (e algoritmo de hash criptográfico para utilização em MACs).

Is resumable - Indica se a sessão pode ser utilizada para novas ligações.

Initialization vectors - Vectores de inicialização (IV) para os modos de cifra simétrica que os utilizam.

Master secret - Chave secreta partilhada por cliente e servidor e da qual são derivados todos os segredos utilizados na sessão (chaves e IVs).

Session identifier - Uma sequência arbitrária de bytes escolhida pelo servidor para identificar a sessão.

X.509 certificate of the peer - Certificado do interlocutor.

Server/Client random - Números aleatórios escolhidos por cliente e servidor para estabelecimento da ligação.

Server/Client write MAC secret - Chaves utilizadas por cliente e servidor para efectuar MACs sobre dados transmitidos.

Server/Client write key - Chaves utilizadas por cliente e servidor para cifrar dados transmitidos.

Sequence numbers - Contadores sequenciais das mensagens enviadas e recebidas.

3.5.1 Handshake Layer

Os parâmetros de sessão e ligação utilizados pela Record Layer são estabelecidos pela Hanshake Layer, por sua vez, as mensagens da Handshake Layer viajam sob o controlo da Record Layer.

No começo de uma ligação TLS assume o seguinte procedimento: Numa primeira fase não há qualquer protecção, sendo utilizado uma cipher spec nula até ao término da primeira negociação.

Salienta-se que a negociação é iniciada pelo cliente, através da mensagem Client Hello à qual o servidor deve responder com uma mensagem equivalente. Ficando assim acordado:

- A versão do protocolo SSL a utilizar
- O identificador da sessão e os números aleatórios
- Os algoritmos criptográficos a utilizar (os mais fortes, suportados)
- O algoritmo de compressão a utilizar

Se no decorrer da ligação houver necessidade de autenticação por parte do servidor, este envia o seu certificado X.509 ao cliente, que o valida. Assim, além da validação habitual o cliente assegura-se de que o nome de domínio do servidor, indicado no certificado, está correcto. Nesta fase, podem também ser enviados parâmetros específicos do servidor para o acordo de chaves (Server Key Exchange), caso o certificado não inclua informação suficiente para esta funcionalidade.

Por seu lado, o servidor também pode pedir autenticação ao cliente, solicitando assim o certificado correspondente (Certificate Request), o qual inclui um desafio para ser utilizado na autenticação do cliente. Esta fase da negociação termina quando o servidor envia uma mensagem Server Hello Done.

Relativamente ao cliente, caso tenha recebido um pedido de certificado tem de enviá-lo ou a negociação falha. Conjuntamente com o certificado o cliente tem ainda de enviar uma assinatura digital do desafio que recebeu, comprovando assim, a posse da chave privada associada ao certificado.

Por fim, o cliente envia os seus parâmetros para acordo de chaves (Client Key Exchange), alterando o seu estado de sessão através do envio de uma mensagem cifrada, a qual indica o seu estado de prontidão (finished). O servidor efectua o mesmo procedimento e a negociação termina tendo sido acordado o Master Secret.

Na figura que se segue, pode-se ver um esquema do modo de autenticação TLS, em que servidor e cliente se autenticam mutuamente.

A autenticação do servidor só ocorre se o protocolo de acordo de chaves implicar a utilização da sua chave privada. Isto acontece sempre que, no protocolo RSAKeyExchange o cliente gera um segredo e cifra-o com a chave pública do servidor para gerar o Master Secret, por sua vez, o servidor decifra o segredo com a sua chave privada.

Nos outros protocolos de acordo de chave que não implicam a autenticação do servidor, os parâmetros públicos utilizados por este, são, também, assinados com a sua chave privada.

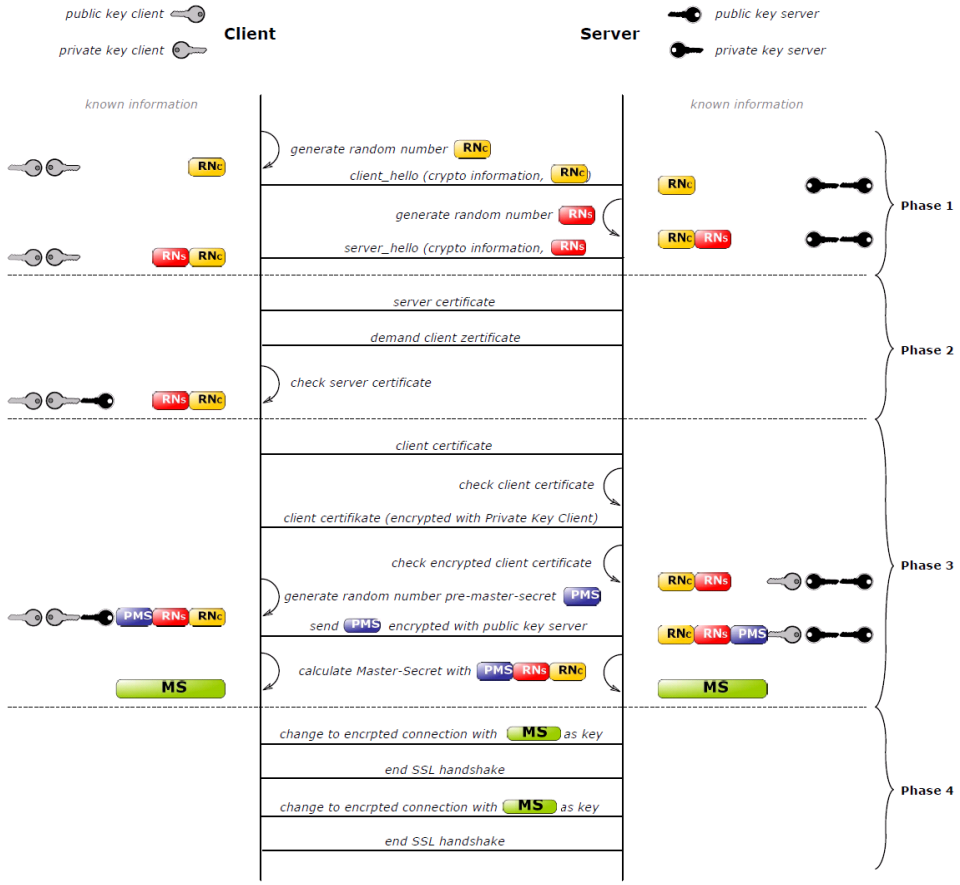


Figura 3.7: TLS four way hanshake

3.5.2 Record Layer

O Record Layer recebe informação arbitrária das camadas superiores, sendo que, os dados são fragmentados em blocos de tamanho variável com um máximo de 214 bytes, denominados SSL Plaintext.

Os blocos TLS Plaintext são comprimidos com o algoritmo da sessão, originando blocos SSL Compressed. Por sua vez, os dados SSL Compressed são protegidos com a cifra e algoritmo de MAC definidos na CipherSpec da sessão (o MAC é calculado antes da cifragem). O resultado é um bloco do tipo TLS Ciphertext.

Estes blocos são permutados entre cliente e servidor, os quais têm de reverter estas transformações por forma obter a mensagem.

3.5.3 Segurança

No que se refere à segurança o TLS é visto como um sistema seguro, considerado uma evolução em relação às versões anteriores, colmatando falhas de segurança importantes.

Relativamente ao SSL um dos problemas mais conhecidos na sua versão 2 do SSL é a vulnerabilidade ao ataque “ciphersuit rollback”, ou seja, um intruso podia editar as mensagens de hello,

trocadas entre cliente e servidor, de forma a que ambos pensassem que o outro apenas conseguia funcionar com um nível de segurança reduzido.

O resto da negociação decorria sem alterações, sendo estabelecida uma ligação com um nível de segurança reduzido, mais vulnerável a ataques por parte do intruso. Este tipo de ataques era possível porque as mensagens de handshake não eram autenticadas. Na versão 3 este problema foi resolvido, obrigando a que todas as mensagens de handshake fossem utilizadas para gerar o valor cifrado nas mensagens finished. No entanto, numa implementação pouco cuidada podem ocorrer outros ataques, nomeadamente "Change cipher spec dropping"

Assim, quando a sessão a ser negociada inclui apenas autenticação, ex. não inclui cifragem, é possível eliminar das mensagens finished a informação de autenticação através da intercepção das mensagens change cipher spec, impedindo assim, a activação da autenticação. Posteriormente fornece-se ao cliente e ao servidor mensagens finished alteradas, estabelecendo-se uma sessão sem protecção. A solução para este ataque consistiu em exigir uma mensagem de change cipher spec antes de uma mensagem finished.

3.6 Identificação de redes Wi-Fi

As redes Wi-Fi utilizam um identificador local, o *Service Set IDentifier* (SSID), para identificar as diferentes redes existentes. Diz-se identificador local, pois só está condicionado a uma determinada área geográfica estando normalmente associado a uma rede específica.

A cobertura dada a uma ou várias estações que estão conectadas a um AP ou um grupo de APs e que partilham o mesmo identificador local (SSID) tem o nome de *Basic Service Set* (BSS).

O ESS caracteriza-se por um ou vários APs que definem o mesmo SSID, ou seja, quando uma ou várias BSS estão interligadas com o mesmo SSID estamos perante um ESS.

Na figura 3.9 é possível verificar que o SSID difundido pelos APs é igual e pertence ao mesmo ESS. No entanto, se a situação verificada na figura 3.8 ocorrer, estar-se-á perante um problema pois as estações não conseguem distinguir os diferentes ESS para o mesmo SSIDs. De facto, esta é uma das limitações que a identificação de redes Wi-Fi possui.

3.7 WEP

O Wired Equivalent Privacy (WEP) foi introduzido pelo IEEE 802.11 original, consistindo na primeira tentativa de se criar um protocolo eficiente de protecção de redes Wi-Fi. Muito criticado por suas falhas, é considerado do ponto de vista de segurança um dos maiores problemas que actualmente existe em redes sem fio. Neste sentido, para garantir a privacidade da rede torna-se necessário recorrer a outros protocolos de segurança (VPN).

Ainda referente a WEP, esta pode ser de 2 tipos:

- WEP - 64 bits (Padrão)
- WEP - 128 bits

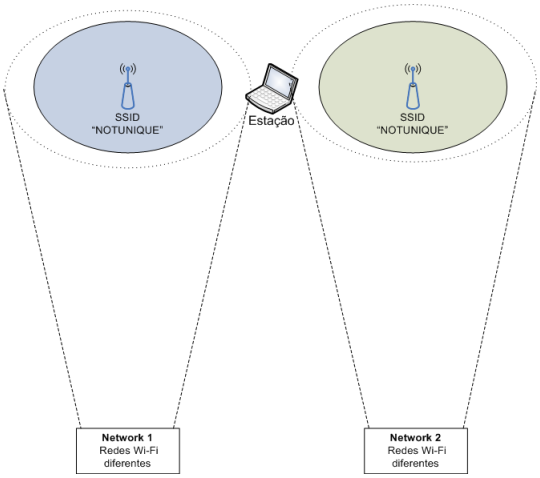


Figura 3.8: Identificação de redes Wi-Fi distintas com o mesmo SSID

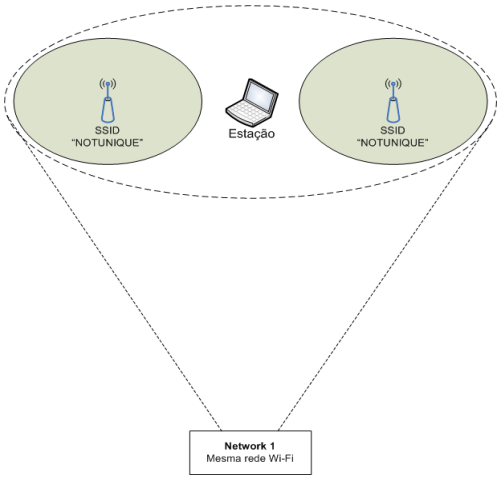


Figura 3.9: Identificação de uma rede Wi-Fi com o mesmo SSID

No que se refere aos modos de autenticação (da estação), existem diferentes modos, sendo eles:

- Sem autenticação - modo aberto (Open mode)
- Com Autenticação - modo partilhado (Shared Mode) (WEP)
  - Desafio/resposta: AP envia desafio, estação devolve desafio cifrado com WEP

O protocolo WEP garante a confidencialidade através da cifragem dos dados, verificando a integridade dos mesmos com o algoritmo CRC32. Passa-se de seguida a enumerar os algoritmos de Cifragem e decifragem que o WEP utiliza.

- Cifragem

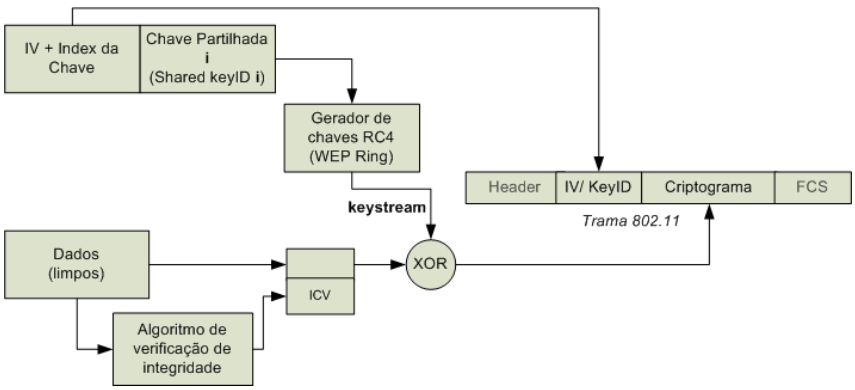


Figura 3.10: Esquema de cifragem WEP

- Descifragem

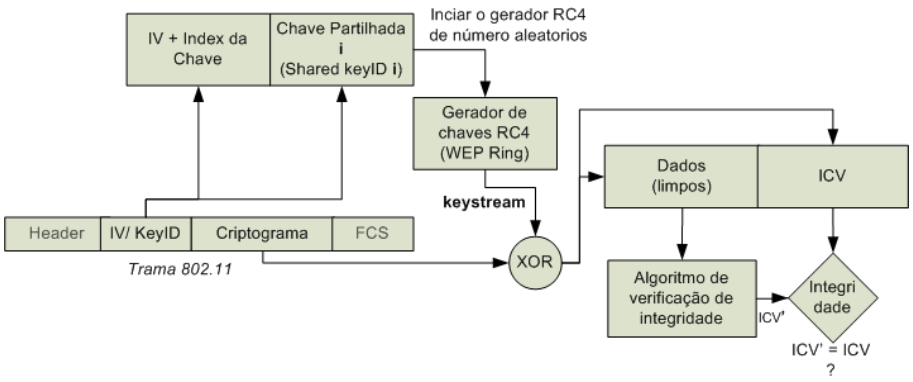


Figura 3.11: Esquema de decifragem WEP

extraída

Embora o WEP tenha inúmeras vantagens também apresenta falhas, nomeadamente:

- O protocolo permite reutilização do seu IV Keystream.
- Através da reutilização de keystream + plaintext attack, é assim possível decifrar dados sem chave WEP
- Tamanho de IV reduzido, mesmo não sendo reutilizado os valores para IV tendem a esgotar-se rapidamente, não estando definidos mecanismos para mudança automática da chave WEP
- IV fracos (Fluhrer et al.) aumenta a possibilidade de descobrir chave WEP
- A Integridade (Integrity Check Value) é baseada em CRC32 (aritmética linear), sendo assim possível alterar a mensagem e o ICV sem que as estações se apercebam
- Autenticação por endereço MAC WEP, não autentica nem garante a integridade do cabeçalho MAC da estação, ou seja, ela pode mudar o endereço MAC e fazer-se passar por outra estação ou AP enviando mensagens de desassociação (ataque DoS)
- Mesma chave WEP para toda a rede - tráfego pode ser escutado/alterado por qualquer estação

Mediante as fraquezas anteriormente apontadas o WEP, vários fabricantes adicionaram outras camadas de defesa, como seja:

- Não são enviados emphbeacons com o SSID
- OAP não responde a pedidos Probe
- As estações têm de saber qual o SSID senão as mensagens são descartadas
- Controlo de acesso pelos endereços MAC das estações (Filtragem de endereços MAC)

3.8 IEEE802.1X

O protocolo 802.1X é um modelo de autenticação universal podendo ser utilizado por todas as redes IEEE 802. A autenticação pode ser mútua sendo efectuada no segundo nível da camada OSI (nível 2).

Possuindo uma grande versatilidade, o protocolo 802.1X permite uma autenticação centralizada, escalável por diversos níveis, ou uma autenticação flexível que use vários métodos de autenticação. No entanto, não é especificado como um protocolo de autenticação próprio, usando para tal, um protocolo de autenticação já existente, o EAP. Desta forma, ganha também a capacidade de distribuição de chaves de encriptação de forma dinâmica. Por exemplo, em autenticações com a cifra de WEP dinâmico é utilizado pelo protocolo 802.11i para as soluções "Enterprise"e permite ainda, a revogação e consequente renovação de chaves utilizadas numa ligação.

Em casos de autenticação mútua ocorre uma troca segura de credenciais, assentando na:

- Autenticação de clientes
- Autenticação do servidor
- Redução da possibilidades de ataques do tipo man-in-the-middle

Requer suporte de software nos suplicantes e no servidor de autenticação (normalmente RADIUS)

Neste sentido, na escolha de suplicantes e servidor deve-se ter em atenção:

- Plataforma a utilizar
- Tipos de EAP suportados
- Proprietário open source
- Repositórios de credenciais suportados (no servidor)
- Custo

Demonstração esquemática de uma autenticação típica de 802.1X:

(I) Situação Inicial, necessitámos de um suppliant, servidor de autenticação (AS) e um NAS, neste caso um AP.

(II) Iniciando-se a autenticação o AP encaminha todo o tráfego para o servidor de autenticação, embora a comunicação ocorra entre a estação e o AP, este último não processa os dados, procedendo, apenas, ao seu encaminhamento para o servidor de autenticação.(Mais concretamente o tráfego vai ser feito entre o AP e o AS em protocolo RADIUS, logo irá cifrado)

(III) Quando a autenticação se dá correctamente, já não é necessário o recurso ao AS e o tráfego da estação utiliza apenas o AP

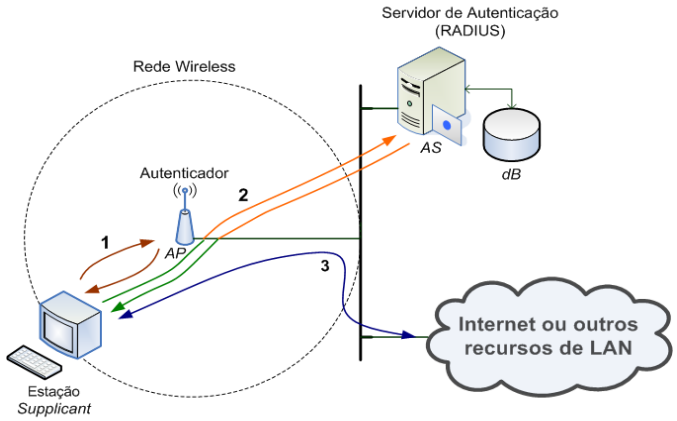


Figura 3.12: 802.1X - Funcionamento geral

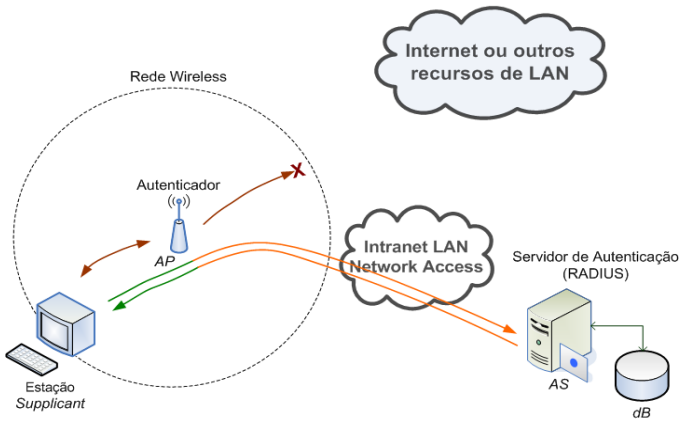


Figura 3.13: 802.1X - Fase de pré autenticação

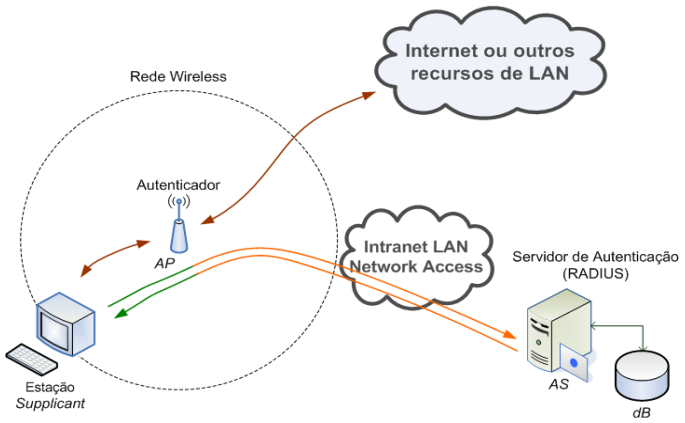


Figura 3.14: 802.1X - Fase de pós autenticação

3.9 IEEE802.11i

Foi aprovada pelo IEEE a norma 802.11i, a qual possui a segurança e a confidencialidade como principal directiva introduzindo melhorias na segurança de redes 802.11. Existia um grande

sentimento de insegurança nas redes baseadas no 802.11, pois estava provado que o mecanismo de segurança WEP era frágil e não protegia o utilizador. Com a introdução desta nova norma passa a existir nos produtos que a implementam uma melhor segurança nos dados.

Esse reforço de segurança assentou, não só, mas também, no aparecimento de novos algoritmos, como seja o CCMP, TKIP, WRAP [25]. Define uma melhor autenticação e Controlo de Acesso, permite também uma melhor gestão dinâmica de chaves. Durante a definição da norma, foram utilizadas noções criptográficas que resultaram na introdução de chaves temporárias e na noção que a chave de autenticação é diferente da chave de cifragem.

Introduz ainda o modo Pre-Shared Key (Personal) e permite utilizar as melhorias do 802.1X

Mecanismo de segurança	**Vantagens e inconvenientes**
Palavra passe - nome de rede - endereços MAC	Simples e universal, mas mau a nível de segurança
WEP	Compatibilidade Wi-Fi, mas mau nível de segurança (excepto, eventualmente, com melhoramentos proprietários)
802.1x	Boa autenticação, mas problemas com o WEP. Melhoramento devido às renovações das chaves
802.11i	Deverá disponibilizar vários mecanismos para melhorar o funcionamento do WEP ou para o substituir (pelo AES). O WPA deverá disponibilizar um 802.11i, leve e utilizável com as placas 802.11b actuais.

Tabela 3.2: Vantagens e Desvantagens dos diferentes mecanismos de segurança

3.9.1 WPA

Todos os produtos que têm como base a especificação 802.11 e quiserem ser reconhecidos como sendo um produto Wi-Fi são regidos pelas regras da Wi-Fi Alliance, a qual, é uma organização sem fins lucrativos que tem como objectivo a interoperabilidade entre os diferentes produtos baseados na norma 802.11. Com a ajuda do IEEE e baseados na 802.11i criaram-se os mecanismos de segurança e confidencialidade WPA e WPA2. Assim, com a introdução da nova norma o algoritmo WEP, que possuía o mesmo propósito, tornou-se obsoleto.

Por sua vez o WPA, é uma solução de compromisso que resolve várias das limitações do WEP. Utiliza o TKIP para a cifragem dos dados sendo que a chave principal não é utilizada para cifrar dados, em vez disso, são utilizadas chaves temporárias derivadas desta. Outra das medidas que permitiu uma maior segurança passou pelo aumento da dimensão do IV(Initialization Vector), assim como, pela substituição do CRC32 (linear) por um mecanismo de hash (MIC).

O WPA permite ainda o uso do 802.1X pois tem suporte para a gestão dinâmica de chaves, mas apenas suporta o modo infraestruturado. Relativamente aos modos de operação este mecanismo suporta o modo Enterprise e o modo Personal (PSK).

Mediante a utilização do algoritmo Temporal Key Integrity Protocol (TKIP), o WPA continua a utilizar o sistema de encriptação RC4 mas disponibiliza um mecanismo de controlo de integridade mais eficaz e uma melhor aliança entre a chave de encriptação e o vector de inicialização. Pode-se

assim dizer que o TKIP permite melhorar significativamente a segurança dos produtos 802.11, com a vantagem de puder ser usado nos interfaces rádio existentes, através da modificação do software incluído (designado por firmware).

O WPA2 é uma solução mais robusta, usa os algoritmos CCMP e TKIP e implementa integralmente a norma 802.11i. Suporta dois modos, sendo eles o modo infraestruturado e o modo Ad-Hoc, suportando ainda dois modos de operação o Enterprise e o Personal (PSK)

Outras das alterações introduzidas por este mecanismo ocorreu no sistema de encriptação por bloco, designado por Advanced Encryption Standard (AES). Esta foi uma modificação profunda na segurança permitindo disponibilizar um sistema bastante mais robusto, sendo, no entanto necessário novos componentes, uma vez que não poderá funcionar com base nas interfaces Wi-Fi actuais (pontos de acesso, principalmente).

Na primeira etapa é feita uma associação 802.11, na segunda etapa são negociadas duas chaves, MSK e EMSK, entre o suplicant e o servidor de autenticação, sendo a MSK transmitida de forma segura ao autenticador. A partir desta chave é calculada a chave PMK. por último esta é usada para negociar uma nova chave, PTK, a partir da qual é obtida a chave TK usada pelo TKIP ou pelo AES-CCMP.

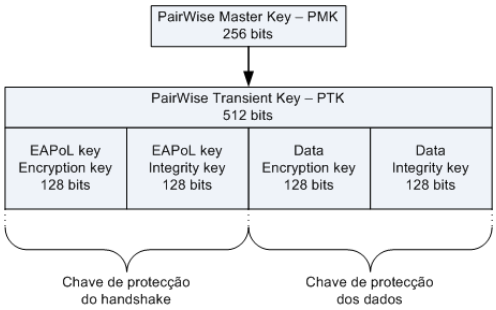


Figura 3.15: Organização da chave PMK/PTK

WPA2, modo de cifra:

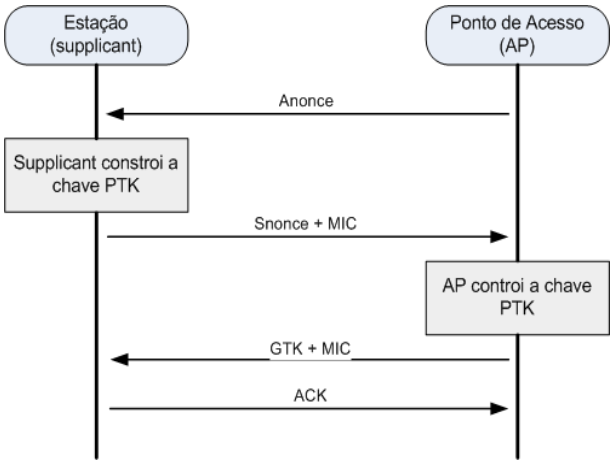
- **Negociação PTK –** A PTK é dividida em três componentes com utilizações distintas:

  KCK (Key Confirmation Key – usada para verificar o MIC;

  KEK (Key Encryption Key) – usada na transmissão de novas chaves temporárias;

  TK (Temporal Key) – usada na cifragem do tráfego.

A integração do mecanismo genérico IEEE 802.1x também representa um grande passo no sentido da melhoria da segurança, uma vez que permite acolher um qualquer sistema de autenticação e mecanismos genéricos de tipo EAP. O mecanismo de autenticação adoptado mais frequentemente nos produtos 802.11 é o EAP-TLS, o qual disponibiliza as funcionalidades de uma autenticação mútua entre a estação 802.11 e o ponto de acesso, além de distribuir as chaves de encriptação de forma dinâmica. Por sua vez, estas chaves são diferentes para cada estação associada ao ponto de acesso. O mecanismo IEEE 802.1x está actualmente integrado na norma IEEE 802.11i e já está presente em muitos produtos comerciais.



- A GTK é alterada periodicamente ou sempre que um dispositivo abandona a rede.
- Quando tal acontece, a estação envia a nova GTK cifrada usando a KEK, sendo-lhe adicionado um MIC, a todas as estações da rede.
- A estação confirma a recepção da novo GTK.

Figura 3.16: Mensagens trocadas numa autenticação WPA2

WPA, embora a sua criação tenha trazido inúmeras vantagens também lhe foram identificadas algumas fraquezas, nomeadamente:

- Passphrase pode ser o elo mais fraco (Modo Personal)
- Permite ataques de força bruta e é um processo lento, mas possível de optimizar com o uso de dicionários ou Rainbow Tables

Com o WPA/WPA2 as redes Wi-Fi ficaram mais seguras, mas muito complicadas para quem quer sistemas de baixa complexidade, no entanto, é impossível ter-se uma segurança igual à do cabo.

3.10 DNS

A infra-estrutura da Internet actual é baseada em IP, mais propriamente em endereços IP ou seja, cada sítio a que o utilizador se liga corresponde a um endereço IP, o qual na maioria das vezes não é conhecido pelo utilizador devido à dificuldade de memorização do mesmo, pois não é facilmente associado à organização que representa ou serviço que presta.

Foi com o objectivo de facilitar a navegação na Internet e a sua flexibilização que em 1983 se criou o DNS, sigla que corresponde ao *Domain Name System* (DNS), recurso utilizado em rede cujo principal objectivo consiste na conversão do nome da máquina no endereço IP, sendo este sistema também responsável pela distribuição e gestão dos respectivos nomes ou domínios.

3.10.1 Zonas e Domínios

O DNS encontra-se organizado por zonas e domínios podendo esses domínios criar subdomínios, dando assim origem a uma hierarquia. No topo da hierarquia só pode existir um domínio designado de “root domain”, é um tipo de hierarquia com um tronco comum centralizado.

Referente ainda aos domínios e subdomínios salienta-se que não existe limite para a sua criação e que os domínios que se encontram acima são hierarquicamente superiores aos domínios inferiores. No entanto, ao contrário do que se poderia pensar a existência do subdomínio não significa necessariamente uma delegação de autoridade. Essa delegação de autoridade ocorre entre Zonas as quais são constituídas por conjuntos de: domínios, subdomínios e máquinas. Convém ainda referir que também as zonas, tal como os domínios, podem ser desmembradas em subzonas sendo que cada nova zona possui a sua própria autoridade (Start Of Authority), a qual é representada no seu servidor principal (NS) de domínio superior, podendo ou não ter servidores secundários.

A definição inequívoca de uma posição de um nó num domínio é chamada de *Fully Qualified Domain Name* (FQDN). Esse nó é único e absoluto, não sendo possível nesse domínio haver um nó com a mesma definição. Por exemplo, temos um nó com o nome "meu", que pertence ao domínio com o nome "suse.pt", o FQDN desse nó será "meu.suse.pt.". Existirão muitos nós com o nome "meu" no mundo, mas apenas este existirá neste domínio.

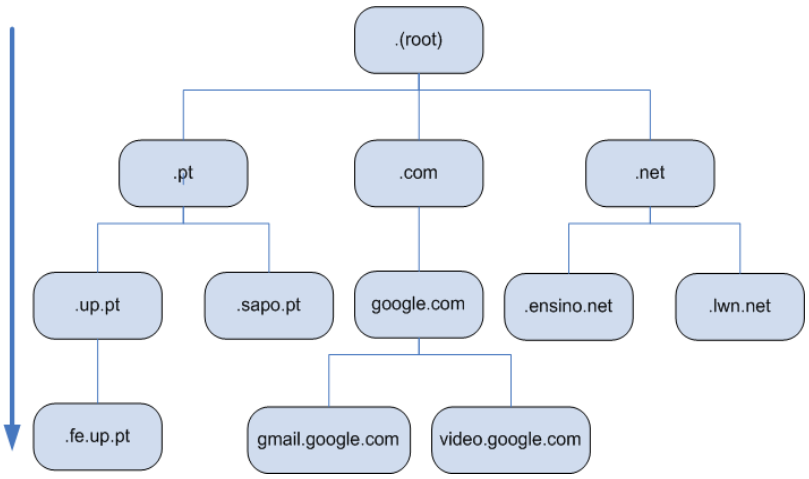


Figura 3.17: Esquema de domínios DNS

3.10.2 Organização do DNS

A informação é armazenada e organizada mediante a necessidade do sistema, ou seja, tendo em atenção o tipo de informação que se pretende guardar, bem como a entidade a que diz respeito. O modo de organização e armazenamento da informação respeita uma série de comandos e siglas também conhecidas como *Ressource Records* (RRs). Os RRs(registos) são usados durante um pedido de DNS, eles permitem descrever os diferentes tipos de dados definidos numa zona enviando-os no formato de texto, cada RR tem uma função específica. De seguida apresenta-se uma tabela com alguns dos RR existentes

Abreviatura	Tipo	Significado
A	IPv4 Address	Relaciona o host FQDN com o seu endereço IPv4
AAAA	IPv6 Address	Relaciona o host FQDN com o seu endereço IPv6
CNAME	Canonical Name	Link de um nome de domínio para o seu destino
MX	Mail Exchange	Relaciona um domínio com os servidores e-mail responsáveis pelo serviço no domínio
NS	Name Server	Nomes do servidor de nome da zona
PTR	Pointer	Nomes das máquinas da zona (usado em *reverse lookup*)
SOA	Start of Authority	Descrição dos parâmetros de gestão de uma zona de autoridade
SRV	Service Location	Registo genérico utilizado para definir a localização de máquinas
TXT	Text	Permite armazenar texto
NAPTR	Naming Authority Pointer	Permite armazenar informação com base em expressões regulares

Tabela 3.3: Tipos de registos do DNS

3.10.3 Resolução

O sistema de nomes baseado em DNS é um sistema hierárquico não centralizado, pois existem 13 "root domains" também conhecidos como mirrors e não apenas um. Quando um utilizador faz um pedido este é redireccionado para o root server que fique mais perto dele, desta forma consegue-se obter o endereço IP pretendido mais rapidamente. Neste sistema distribuído, ou seja, através da delegação de autoridade a informação sobre máquinas e domínios encontra-se distribuída por milhares de servidores de nomes existentes, os quais alteram com frequência a estrutura de domínios, máquinas e zonas de autoridade. Sendo por isso importante definir métodos que permitam descobrir determinado IP nesta hierarquia de domínios e servidores.

Ao processo de tradução de um nome de domínio em um endereço IP dá-se o nome de resolução (lookup). No entanto, actualmente o DNS não trabalha apenas com endereços de IP e nomes de hosts, sendo que esse processo ocorrerá também sobre outro tipo de informação.

Neste seguimento de ideias o *resolver* corresponde à entidade a que as aplicações recorrem quando necessitam de resolver alguma informação no DNS. No decorrer do processo de resolução, a aplicação não intervém, ou seja, a entidade *resolver* comunica directamente com os servidores de nome.

Os métodos de resolução podem ser de três tipos: método iterativo, que como o próprio nome indica recorre iterativamente a vários servidores de nomes de forma a obter a informação desejada; método recursivo, o qual recorre ao servidor de nomes preferidos para obter a informação pretendida, ou seja, espera-se obter uma resposta e não uma referência, sendo este método o mais utilizado apesar de ser o que mais sobrecarrega os servidores de nomes da cada ISP; por último o método de resolução inversa *(reverse lokkup)*, que mapeia o nome da máquina ou FQDN ao endereço IP, desta forma é possível saber nome da máquina para um dado endereço IP.

Resolução Iterativa

Numa resolução iterativa o *resolver* contacta cada servidor de DNS até concluir a resolução do nome pedido. Cada servidor contactado dará uma resposta ao pedido sem contactar qualquer outro servidor de DNS. O processo é repetido até o *resolver* contactar um servidor de DNS autoritário que lhe dê uma resposta positiva ou não consiga dar informações para continuar a iteração.

Durante a procura do domínio que contém o endereço para um determinado nome ou FQDN, as respostas podem ser de vários tipos:

- *Authoritative*, é uma resposta positiva e autoritária, ou seja, o servidor DNS que foi contactado é o que tem autoridade sobre o domínio pedido.
- Positiva, estas respostas dão-se quando o servidor consegue responder com sucesso à pergunta efectuada, podendo ela ser um RR ou um conjunto de RRs.
- *Referral*, ocorre sempre que o servidor de DNS não consegue dar uma resposta positiva ao *resolver*, no entanto, na resposta pode incluir tipos de registos que contribuam para o processo de resolução.
- Negativa, este tipo de respostas dão-se quando o servidor DNS autoritário da zona especificada pelo *resolver* não encontra nenhum registo para aquele nome ou tipo.

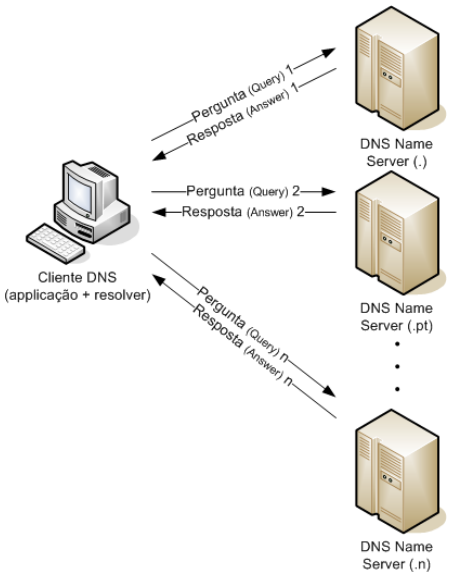


Figura 3.18: Resolução iterativa DNS

Resolução Recursiva

A resolução recursiva recorre a um servidor de DNS (normalmente do ISP) para a obtenção do endereço a que corresponde o nome pedido pelo *resolver*. Neste tipo de resoluções o cliente DNS pede ao *resolver* a resolução de um nome e recebe uma resposta, seja ela positiva ou não. O

resolver é que fica encarregue de perguntar aos diferentes servidores de DNS das diferentes zonas, qual o endereço IP para o nome especificado, ou seja, fica o resolver encarregue da resolução "iterativa".

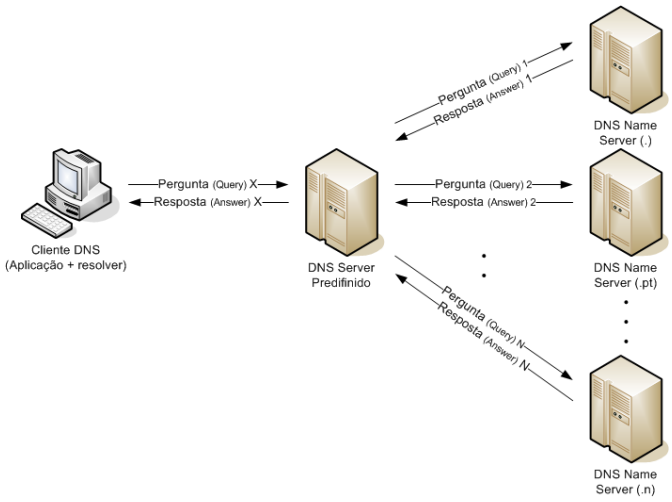


Figura 3.19: Resolução recursiva DNS

Resolução Com Cache

Após a descrição dos métodos usados na resolução DNS os quais eram ineficientes e provocavam sobrecarga nos servidores. Assim foi introduzida uma melhoria substantiva no processo de resolução com a introdução da cache. A cache é um mecanismo que permite a cada uma das entidades intervenientes na resolução guardar durante um determinado tempo útil, a informação obtida (resolvida) de cada um dos servidores de nomes contactado. Neste seguimento de ideias o processo de resolução segue outros procedimentos, o resolver contacta o servidor de DNS predefinido pedindo a resolução de um nome, o servidor de DNS contactado primeiro verifica se na cache interna contém algum registo para aquele nome e só se não constar nenhum registo na cache é que é contactado o outro servidor de DNS autoritário.

Com a introdução da cache houve necessidade de se complementar o serviço DNS com a introdução de dois novos conceitos, assim em complemento ao servidor de nomes primário temos:

servidor DNS secundários , contêm uma cópia dos registos do servidor primário (principal) que periodicamente são actualizados. Estes servidores secundários são também eles "autoritários" para a mesma zona que o servidor DNS primário.

servidor DNS de cache , servidores intermédios que armazenam toda a informação que é recebida durante o processamento dos diferentes pedidos de resolução de nomes. A introdução deste servidores vêm diminuir de um modo geral os pedidos de resolução aos servidores de DNS autoritários. Os servidores de DNS de cache não são autoritários.

Com todas as modificações necessárias aos servidores de DNS foi também reformulada a estrutura das zonas surgindo assim novos parâmetros:

***Time To Live* (TTL)** , é o tempo por defeito que os registos permanecem válidos em cache depois de terem sido disponibilizados por um servidor primário ou secundário.

Retry interval , indica o intervalo de tempo no qual o servidor de DNS secundário tenta reconectar-se novamente após ter falhado.

Expires after , define o tempo máximo a partir do qual o servidor de DNS secundário deixa de responder, se não conseguir contactar o servidor primário.

Refresh interval , é o intervalo entre actualizações obrigatórias do servidor de nomes secundário e o servidor de nomes primário.

Serial , é um número que indica o número de vezes que a base de dados (zona) foi alterada. Através deste parâmetro é possível controlar facilmente se os registos entre o servidor primário e o servidor secundário estão desactualizados.

```
$\gls{TTL} 86400
$ORIGIN porto.pt.
@               IN SOA    ns01.porto.pt. root.porto.pt. (
                                        2008081601      ; serial
                                        3H              ; refresh
                                        15M             ; retry
                                        1W              ; expiry
                                        1D )            ; minimum

..
..
```

Resumindo, a cache DNS guarda localmente os resultados das requisições de resolução de nomes para utilização futura, evitando a repetição de pesquisas e aumentando drasticamente a velocidade de resposta. Essas informações são armazenadas pelo período de tempo determinado no *Time To Live* (TTL), que é um valor enviado junto da resposta de cada pedido.

Resolução Inversa

Se se recordam o DNS nasceu da necessidade de se obter o endereço IP para um determinado nome, isto porque a maioria do tráfego é feita em TCP/IP. Com a evolução da Internet tornou-se importante obter o resultado inverso, ou seja, para um dado IP qual o nome associado. Assim nasce a resolução inversa que tem como objectivo complementar o serviço de DNS.

A adaptação do serviço de DNS para a resolução inversa foi complexa, pois um dos grandes problemas estava relacionado com o facto do nome de domínio ser escrito pela ordem contrário ao endereço IP. Isto é, o domínio hierarquicamente superior está escrito à direita nos nomes contrariamente ao endereço IP que é à esquerda. Assim, houve a necessidade de criar duas zonas distintas, uma para a resolução directa (normal) e outra para a resolução inversa. Além dessas alterações

foi também criado o registo PTR (Pointer Records) que é o registo que associa a informação do endereço IP ao nome.

O pedido PTR é feito invertendo o endereço IP de forma a que o processo de resolução inversa tenha a mesma lógica que a resolução directa. Sendo assim, têm-se os pares de endereços menos significativos primeiro e só depois os mais significativos. Foi ainda necessário diferenciar um endereço IP de um nome, adicionando o sufixo 'in-addr.arpa'. Desta forma o servidor de DNS sabe a que zona recorrer para efectuar a resolução inversa.

Dando um exemplo, se se desejar obter o nome de uma máquina com o endereço IP 172.16.10.2 o *resolver* deve converter o endereço 172.16.10.2 para 2.10.16.172.in-addr.arpa.

```
$\gls{TTL} 86400
$ORIGIN 10.16.172.in-addr.
..
..

                        IN NS   ns01.porto.pt.

3                       IN PTR  \gls{RADIUS}.porto.pt.
5                       IN PTR  mail.porto.pt.

;ZSK Key
$INCLUDE /var/bind/porto.pt+005+32454.key
;KSK Key
$INCLUDE /var/bind/porto.pt/ksk/Kporto.pt.+005+46414.key
```

3.10.4 Dinamic DNS

Tradicionalmente as zonas de DNS são actualizadas manualmente pelos administradores de rede, situação bastante prática numa rede estática e pouco polimórfica. No entanto os hosts podem assumir endereços temporários (DHCP), os quais podem ser diferentes ao longo do tempo. Como resultado o número e a frequência de mudanças dos endereços IP torna impraticável a administração manual do servidor de DNS. Para resolver o problema a IETF melhorou o serviço de DNS tornando-o mais dinâmico, criando o que é agora conhecido como *Dynamic Domain Name Server* (DDNS), mecanismo para a gestão da identidade de hosts de redes dinâmicas, termo que engloba o processo de:

- Actualização dinâmica
- Notificação
- Tranferência de zona incremental (IXFR)

As três novas extensões desenvolvidas são inter-dependentes e permitem flexibilizar de grande modo a administração e a usabilidade do DNS.

3.10.5 Vulnerabilidades

Nesta sub-secção abordar-se-ão as vulnerabilidades a que o protocolo de DNS se encontra sujeito. Nomeadamente a falta de autenticidade dos registos, os quais podem ser enviados por um servidor forjado, bem como a falta de integridade, podendo esses mesmos registos ser alterados sem que o utilizador tenha conhecimento. Salienta-se ainda que os dados falseados/inválidos podem não só quebrar a confidencialidade das comunicações mas também provocar ataques de indisponibilidade dos serviços (*Denial of Service* (DoS)).

Neste sentido, identificam-se de seguida os possíveis ataques que exploram as fraquezas do DNS.

Ataque por buffer overflow , é um ataque complexo que explora as falhas na programação do software, possuindo como objectivos a obtenção de acesso à linha de comandos do servidor de nomes ou permissão para modificar os ficheiros das zonas. Neste tipo de ataques são exploradas falhas no código ou no executável de modo a corromper a stack (memória), correndo de seguida um código malicioso, previamente enviado, que possibilita o acesso à máquina, nomeadamente o acesso a um porto (socket) ou a algo mais complexo.

Ataque para obtenção de informação (Information Disclosure) , correspondem a ataques que ocorrem sobre a informação dada durante uma transferência de zona, permitindo ainda obter a versão do ficheiro da mesma. Este tipo de ataques pode não ser nocivo, mas revestem-se de alguma importância, ou seja se um atacante tem acesso a informações sensíveis sobre a configuração interna da rede, fica numa situação privilegiada para poder atacar a infra-estrutura com sucesso. Isto porque, se a transferência de uma zona for bem sucedida podem lá constar informações que indiquem por exemplo, nome de projectos especiais, identidades de máquinas ou informações sobre o tipo de software que nelas está instalado.

Ataque por DOS (Denial of Service) negação de serviço , o objectivo deste ataque é deixar o servidor de nomes de tal forma "baralhado" que deixe de providenciar serviço, ficando inoperacional. O serviço de DNS usa essencialmente UDP para o transporte dos pedidos e respostas do sistema DNS, permitindo dessa forma que nunca seja completado um circuito que permita identificar o autor de um pedido, tornando-se assim quase impossível de rastrear e bloquear os ataques de DOS. Por outro lado, para a criação de um ataque de DOS basta que sejam produzidos uma quantidade maior de pedidos dos que aqueles que o servidor de nomes consegue resolver, desta forma o servidor irá ficar cada vez mais lento até não conseguir responder aos pedidos legítimos.

Ataque por envenenamento da cache (cache poisoning) , é um ataque difícil de alguma complexidade, no qual a cache do servidor de nomes é deliberadamente contaminada por um atacante. Para isso, o atacante tenta adivinhar a transacção dos pedidos de DNS através de sofisticados métodos de predição ou por recurso sucessivo dos pedidos. Este tipo de ataque é dos mais perigosos e explora uma funcionalidade do DNS, a cache, podendo ser separado

em dois níveis de gravidade: o nível mais baixo consiste em adicionar informação adicional aos pedidos de DNS, ou seja, o atacante inclui informações falsas nos registos retornados pelo servidor de nomes legítimo. Para tal o atacante usa a secção Additional da resposta e inclui registos que ele deseja associar a determinados nomes, sendo o CNAME, NS e DNAME os mais afectados. O nível mais grave consiste na substituição de uma zona inteira na cache. Para que este tipo de ataque possa ocorrer são necessário vários IP's ou pelo menos um gerador de pedidos de DNS que permita mudar o endereço de origem. De seguida são enviados diversos pedidos de DNS ao servidor de nomes alvo (ex: ns1.sa.com) sendo que todos eles apontarão para o mesmo domínio (ex:www.google.com). Assim, o servidor de nomes alvo irá então ser obrigado a mandar vários pedidos de DNS ao NS (Name Server) do google.com, sendo que cada um desses pedidos irá ter um ID único (ID de transacção) e será processado em separado e paralelamente. Desta forma o servidor de nomes alvo irá estar a espera de diversas respostas àqueles pedidos, nesse momento o atacante envia milhares de respostas para o servidor de nomes alvo tentando acertar qual o ID da transacção, assim como o porto que enviou o pedido (pode também tentar adivinhar o endereço, mas neste caso, as possibilidades encontram-se reduzidas a quase nada, pois a entropia conseguida [porto + ID + IP] é muito alta para em tão pouco tempo se conseguir enganar o servidor de nomes). Se o ataque for bem sucedido o servidor de nomes fica com um registo falso, apontando assim os utilizadores para um sítio na Internet adverso. O registo falso estará na cache por um período nunca maior do que o seu valor de TTL.

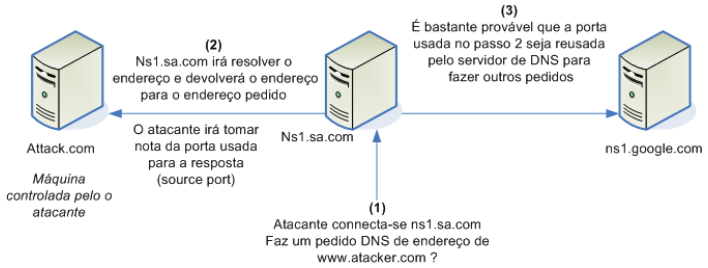


Figura 3.20: Fase de estudo do ataque ao servidor de DNS

Ataque por DNS hijacking , também conhecido por Man-in-the-middle atack, consiste no envio, por um atacante, de registos DNS corrompidos para uma estação alvo que quando aceites pela estação permitem encaminhar a mesma para outros locais que não os desejados. Neste tipo de ataque explora-se o facto do DNS usar pacotes UDP que não permitem verificar a origem, bem como o facto de não usar nenhum tipo de autenticação não lhe sendo por isso permitido verificar a integridade ou autenticidade dos pacotes recebidos. Os passos necessários para a execução de um ataque por DNS hijacking podem ser vistos na figura 3.21, na qual o atacante espera que a estação mande um pedido de DNS retirando desse pedido o endereço de IP do servidor de nomes, o porto de destino e o porto de origem, assim como o ID da transacção. Deste modo ele cria um pacote de DNS de resposta que iguala os parâmetros do pedido e envia-o para a estação alvo. Esta operação é normalmente rápida

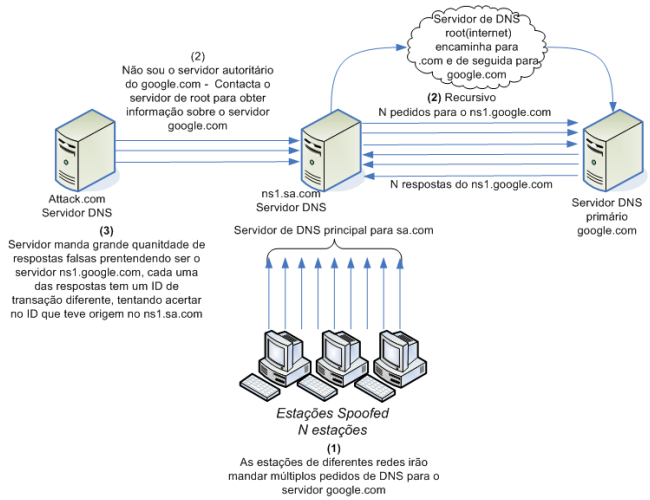


Figura 3.21: Esquema de um ataque ao DNS usando o método de cache poisoning

sendo que, na maioria dos casos, a resposta do atacante chegará primeiro que a resposta do servidor de nomes legítimo. Por sua vez, a estação não sabe se o pacote é de confiança ou não mas como cumpre todos as regras é aceite, a partir desse momento o atacante pode dar informações falsas ao cliente sem que ele se aperceba.

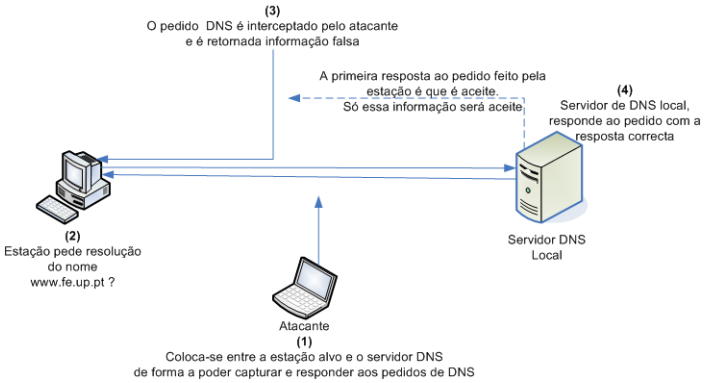


Figura 3.22: Esquema de um ataque ao DNS usando o método MIDM

Por último, existe um tipo de ataque que é mais um alerta que propriamente um ataque e ocorre em redes que tenham servidores de nomes partilhados. Nestes casos o mesmo servidor de nomes é partilhado por diversos utilizadores e se for permitido criar domínios *on-demand*, um utilizador registado pode abusar da confiança que lhe foi dada e criar um ficheiro de zona que lhe permita adicionar o domínio ex:www.google.com e preencher correctamente os diversos registos A e MX. Desta forma todos os clientes que tenham o seu servidor de nomes principal a apontar para o servidor de nomes partilhado vão incorrer no risco de obterem informação falsa. Pode-se dizer que é mais uma falha na arquitectura ou na implementação da segurança de zonas que uma vulnerabilidade do sistema de DNS.

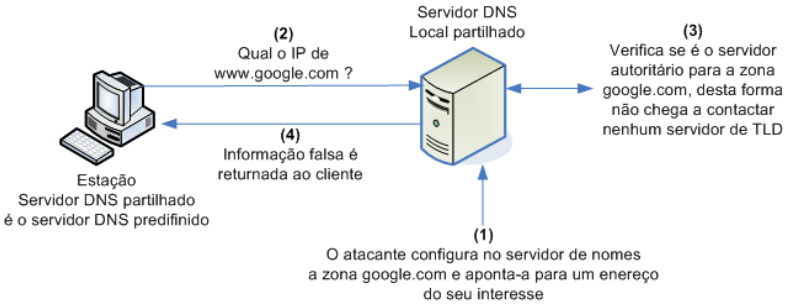


Figura 3.23: Esquema da vulnerabilidade do servidor de DNS partilhado

3.11 DNSSEC

Com os constantes problemas que surgem no DNS, começa-se a registar um crescimento na adopção do DNSSEC. O *Domain Name System Security Extensions* (DNSSEC) também conhecido como 'DNS seguro' é um conjunto de especificações (extensões) que visam resolver as várias falhas detectadas no DNS. Estas extensões têm por fim proteger e autenticar o tráfego de DNS, não estando no seu âmbito de acção a confidencialidade desses registos. A validação dos mesmos é feita através de assinaturas digitais as quais são baseadas em chaves criptográficas. Para tal o DNSSEC usa a criptografia simétrica para a assinatura de RRs e a criptografia assimétrica para operações de transferência de zonas entre os servidores de nomes. Para garantir a autenticidade das informações recebidas o DNSSEC permite:

- Verificar a origem dos dados autenticados
- Verificar a integridade dos dados
- Verificar a não existência de um RR de forma autenticada

Para que estas novas funcionalidades fossem adicionadas ao sistema de nomes foi necessária a inclusão de novos registos (RRs), nomeadamente:

DNSKEY – contém as chaves associadas a um determinado RR, incluindo ainda as chaves públicas que assinam digitalmente a zona.

NSEC – permite identificar quais os FQDNs válidos para uma determinada zona. Este registo é usado na verificação da não existência de um FQDN/endereço numa zona.

NSEC3 – persegue os mesmos objectivos que a NSEC, no entanto permite evitar o problema da enumeração de nomes do registo NSEC.

RRSIG – contém a assinatura dos diferentes tipos de RRs existentes na zona incluindo ainda os RRs DNSKEY e NSEC. Assim, para cada RR para o qual a zona é autoritária terá de existir o registo RRSIG correspondente.

DS (Designated Signer) , este registo contém as *hashs* das chaves públicas das zonas inferiores delegadas (child zones). Sendo essenciais para a criação da cadeia de confiança, princípio básico em que o DNSSEC se baseia.

3.11.1 Registos

Após a identificação dos diversos registos adicionados proceder-se-á a uma descrição mais detalhada dos mesmos, bem como da sua estrutura:

DNSKEY contém as chaves públicas associadas à zona, podendo ainda associar outras chaves a determinados registos da zona. Estes registos são, regra geral, constituídos pelas chaves *Zone Signing Key* (ZSK) e *Key Signing Key* (KSK) o tamanho das chaves é variável e depende do compromisso entre a longevidade da chave e o poder de processamento disponível para as processar. O registo possui um campo que identifica o tipo de chave, o protocolo para que a chave é usada, o algoritmo utilizado pela chave e a chave.

***Key Signing Key* (KSK)** – É a chave que assina chaves possuindo uma entropia de tamanho superior às chaves das zonas, também responsável por assinar todas as chaves do domínio.

***Zone Signing Key* (ZSK)** – Esta chave é utilizada para assinar cada zona de forma individual, sendo o seu tamanho inferior à chave KSK.

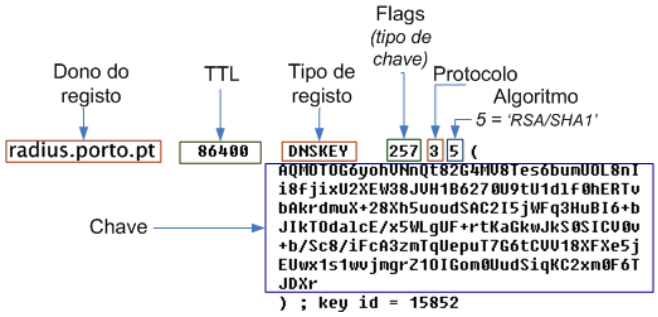


Figura 3.24: Esquematização dos elementos do DNSKEY

DS – Este registo contém o resumo da chave pública (DNSKEY) de uma zona o que indica que a zona delegada é assinada digitalmente. É usado na autenticação entre as zonas delegadas (child zones) e zonas primárias (parent zones), aparecendo apenas nos servidores destas últimas, pois, baseia-se nas chaves de DNSKEY das zonas delegadas. O registo DS, para além do resumo é também constituído pela identificação do algoritmo do mesmo e pelo seu tamanho.

NSEC (Next SECure) – registo que tem como objectivo provar a inexistência de um domínio ou tipos de registo desse domínio. Na sua resposta é dado um intervalo de nomes válidos no qual se verifica se o nome é inexistente, resolvendo-se assim o problema das respostas

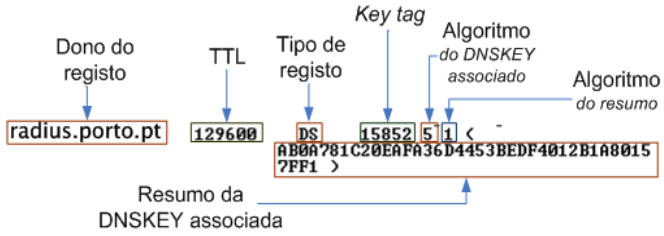


Figura 3.25: Esquematização dos elementos do Zona DS

negativas. O NSEC define o intervalo entre dois nomes de zona consecutivos indicando qual o próximo registo, facto que obriga a que durante o processo de assinatura a zona seja ordenada de forma canónica (em ordem alfabética), bem como seja inserido um registo NSEC por cada nome. O registo NSEC é constituído pelo nome do próximo registo, assim como pela lista de registos existentes para o actual nome.

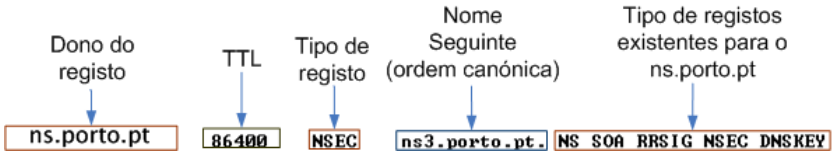


Figura 3.26: Esquematização dos elementos do NSEC

NSEC3 (Hashed Next SECure) – este registo é uma revisão do registo NSEC realizada de forma a evitar ataques que permitissem a divulgação de informação o que no caso do NSEC foi conhecido como *zone walking*. Neste sentido, o registo NSEC3 utiliza um resumo do seu nome para indicar o próximo registo permitindo que seja concatenada uma 'semente' (salt) em cada nome, aumentando a sua entropia e segurança. Essa 'semente' segue juntamente com o registo de maneira a que os resolvers ou outros servidores de nomes o possam validar. Este registo é constituído pelo algoritmo do resumo, número de iterações do algoritmo, número de registos existentes, semente (salt), resumo do próximo nome e pelo número de registos que o nome possui na zona. Foi ainda adicionado um novo registo de controle chamado NSEC3PARAM, o qual será aprofundado de seguida.

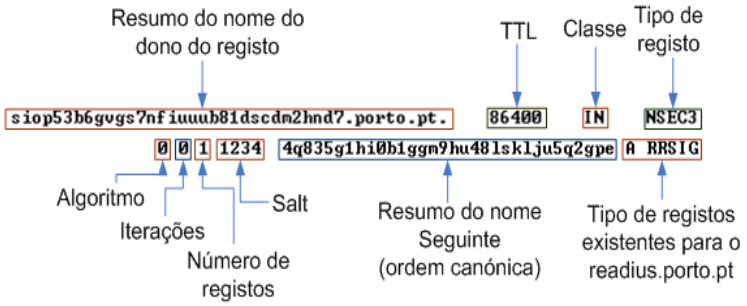


Figura 3.27: Esquematização dos elementos do NSEC3

NSEC3PARAM (NSEC3 Parameters) – este novo registo de controle é destinado, especificamente, ao servidor de nomes autoritário da zona, servindo-lhe de referência, pois permite-lhe escolher a resposta negativa correcta a enviar. Possui como principal função indicar ao servidor como deve calcular os resumos dos nomes, sendo por isso adicionado na parte mais baixa da zona (em termos canónicos) durante o processo de assinatura.

Figura 3.28: Esquematização dos elementos do NSEC3PARAM

RRSIG – registo que corresponde à assinatura digital de outro registo (utilizando a chave privada do servidor de nomes) tornando possível que o resolver autentique os registos de uma zona. Assim, para cada registo teremos um RRSIG correspondente, pois só dessa forma é possível averiguar a autenticidade e integridade de um qualquer registo. O RRSIG é constituído pelo tipo de registo que assina, pelo algoritmo usado para a assinatura, pelo número de etiquetas, pelo TTL original do registo, pelo intervalo de validade da assinatura, o qual contém a data de início e a data de fim (incluindo as horas e minutos), pela semente da chave usada (Key Tag), pela zona e por último pela assinatura.

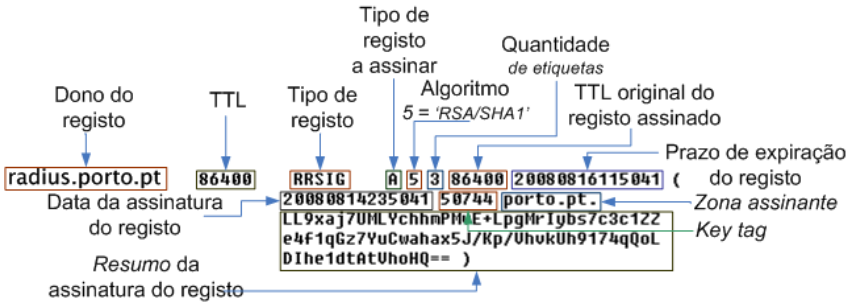


Figura 3.29: Esquematização dos elementos do registo de RRSIG

O processo de assinatura de uma zona é feito recorrendo à chave privada associada à mesma, pois a utilização da chave pública permite apenas verificar a autenticidade dos registos. Tendo em conta o acima referido se um domínio se encontra devidamente "assinado", implicando isso todos os registos afectos a esse domínio, então um servidor de nomes externo (ou resolver) pode verificar a autenticidade das respostas que obtém utilizando para tal a chave pública do servidor de nomes, verificando assim se efectivamente o registo foi assinado por aquela chave, protegendo deste modo o utilizador e todo o sistema dos diversos tipos de ataques.

3.11.2 Cadeia de Confiança

Apesar dos novos registos incluídos no DNSSEC, já detalhadamente explicados na sub-secção anterior 3.11.1, ainda não ficou explícito o processo de validação do mesmo ao qual é dado o nome de cadeia de confiança, sendo por isso, de seguida, explanado em maior detalhe os passos necessários para a execução dessa validação. Neste sentido a validação entre os diferentes domínios é feita recorrendo às chaves públicas (mais concretamente ao seu resumo), para isso e recordando que é no registo DNSKEY que cada zona segura possui a sua chave pública, esta para ser certificada por uma zona superior (parent zone) recorre ao registo DS. Assim, quando o par de chaves é gerado nas zonas delegadas (child zones) elas enviam o resumo da sua chave pública para a sua zona primária (parent zones) para que esta as certifique inserindo para isso o resumo (DS) na sua zona, validando desta forma a chave pública da zona delegada (child zone). O mesmo processo é efectuado em todos os servidores de nomes até serem atingidos os servidores de nomes raiz (TLD), ou seja, o resumo das chaves públicas das zonas será configurado em todos os servidores de nomes e cada servidor poderá certificar-se se as assinaturas que lhe foram enviadas são autênticas. Todo este processo é designado de cadeia de confiança o qual é desencadeado pela verificação dos diferentes registos de DNSKEY, seguido dos registos DS da zona primária correspondente e por último pela verificação de uma chave pública segura. Assim, os resolvers e servidores de nomes necessitam apenas de possuir a chave pública do servidor de TLD (root) para poderem validar todos os registos de DNS recebidos. O acto de introdução da chave pública nos resolvers designa-se por *Secure Entry Point* (SEP) o qual permite inserir uma ”âncora” (anchor keys) no resolver para que este estabeleça a cadeia de confiança. Numa situação ideal o SEP referenciado é a chave pública dos servidores de TLD, sendo assim, apenas é necessário ter uma chave no resolver de cada cliente.

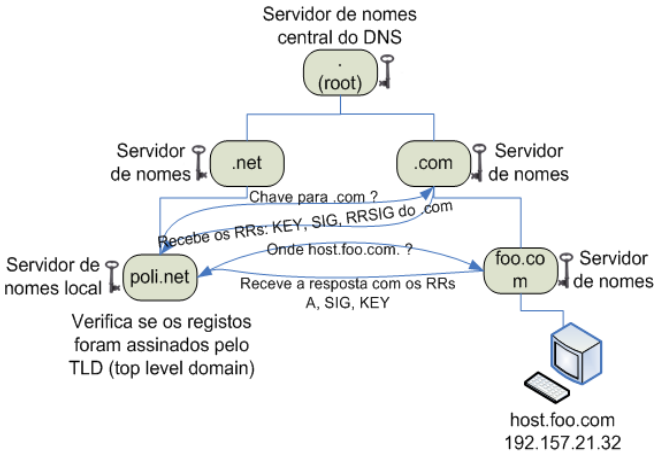


Figura 3.30: Esquema de uma ilha de confiança com o SEP no domínio .COM

O SEP pode ser definido em outros domínios que não os TLDs para isso, basta inserir nos resolvers ou servidores de nomes a chave pública do domínio que se deseja, obtendo-se desta forma a cadeia de confiança necessária para validar os registos. No entanto, esta medida possui como inconveniente o facto de para cada domínio que se deseja validar ser necessário a inserção da

sua chave. A todo este conceito designa-se por ilhas de confiança *(islands of trust)*, pois apesar de todos os domínios não estarem 'seguros' pelo DNSSEC consegue-se garantir que todas as zonas delegadas pelo servidor de nomes, do qual se introduziu a chave pública no resolver, estejam seguras.

3.12 IPTables

O iptables é um programa que permite ao utilizador controlar uma *framework* de pacotes que vem embutida no kernel da sua máquina (pressupõe-se a utilização do SO linux). Essa framework pode ser utilizada como uma firewall e não se resume a filtros de pacotes sendo antes um sistema integrado composto por filtros de pacotes, filtros de estados, Intrusion Detection Systems (IDS), proxies, etc, utilizados pela estação para a sua protecção. Neste sentido é errado pensar que uma máquina que utilize o iptables esteja completamente protegida. Isso porque, o iptables é apenas um filtro de pacotes e estados, logo só analisa os dados contidos nos cabeçalhos IP dos pacotes que chegam à máquina. Assim pode-se definir o iptables como um dos principais componentes de uma firewall mas não o único, este opera sobre o layer 3 e o layer 4 da camada *Open Systems Interconnection* (OSI). O princípio de funcionamento da firewall é ilustrado no esquema 3.31 mediante o qual se pode ver o caminho seguido pelos pacotes desde o momento que são processados pelos módulos do kernel (sistema) até à finalização de todo o processo.

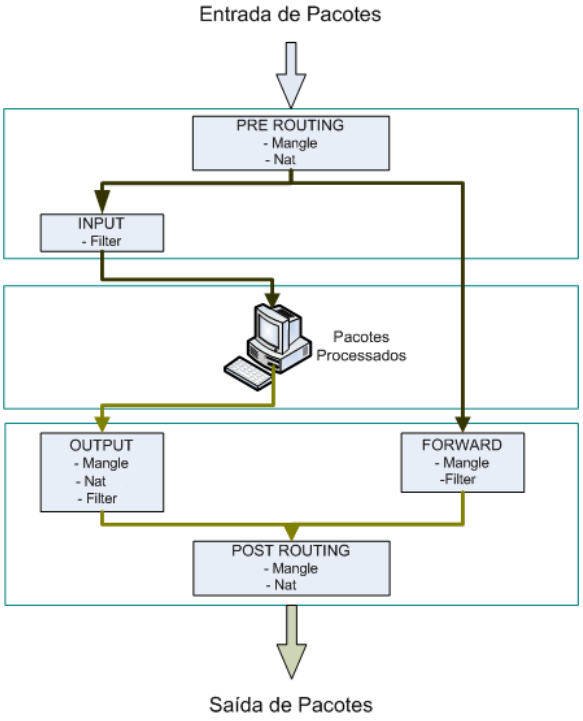


Figura 3.31: Esquema básico do processamento das regras IPTables

Com o iptables o utilizador consegue controlar o tráfego da sua máquina de forma eficaz, recorrendo para isso a regras no tratamento dos pacotes. Todo o conceito de regra, assim como o

seu uso é descrito de seguida na sub-secção de regras.

3.12.1 Regras

Uma regra estabelece os parâmetros de tratamento de um determinado pacote, permitindo que o iptables realize uma determinada acção (como descartar ou permitir a aceitação de um pacote) de acordo com: o endereço e/ou porto de origem, o endereço e/ou porto de destino e ainda o interface de origem e/ou de destino. Outras das suas funcionalidades consiste na definição de parâmetros mais avançados para o tratamento de protocolos específicos, como o TCP ou UDP, entre outros. As regras são armazenadas dentro das *chains* e processadas de acordo com a ordem em que foram inseridas. São dois os métodos que permitem adicionar as regras, nomeadamente o método *input* (-I) que coloca a regra no topo das regras a serem processadas, ou o método *append* (-A) sendo que, neste último as regras são adicionadas no fim da *chain*. Por outro lado, também é possível apagar regras inseridas, para tal recorre-se ao comando (-D) para apagar as regras uma a uma ou ao comando (-F) *flush* para apagar todas as regras existentes. Como foi referido anteriormente as regras são armazenadas no kernel, infelizmente não existe nenhum método no kernel que permita guardar essas regras, assim sempre que a máquina for reiniciada as regras também serão. Porém, existem programas que permitem obter e gravar as regras definidas no iptables do kernel, bem como restaurá-las, são eles o iptables-save e iptables-restore respectivamente.

São necessários definir alguns parâmetros para a criação de regras, os quais serão enumerados de seguida, salienta-se, no entanto, que nem todos são obrigatórios:

– Interface: Definição do interface ao qual se deseja aplicar a regra;

– Protocolo: Definição do protocolo (ou todos) ao qual aplicar a regra;

– Módulo: Definição da listagem de módulos disponíveis para utilização;

– Ip Origem: Definição do endereço IP de origem do pacote;

– Porto Origem: Definição do porto de origem do pacote;

– Ip Destino: Definição do endereço IP de destino do pacote;

– Porto Destino: Definição do porto de destino do pacote;

– Identificador: Definição do tipo de pacote a filtrar. Exemplos: ! -syn, -state INVALID,

– icmp-type: Definição do tipo de resposta ICMP ex: echo-reply

A utilização mais comum das regras consiste na restrição de uma ou mais máquinas a um IP ou porto, podendo no entanto a sua utilização ser mais abrangente.

3.12.2 Chains e Tabelas

A estrutura lógica onde são adicionadas as regras designa-se por *chains*, existindo dois tipos: as *chains builtin* (por defeito) e as chains criadas pelo utilizador (root). As primeiras, do tipo *builtin*, contém as *chains* INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING, as quais estão sempre presentes e não podem ser apagadas, outra das suas particularidades prende-se com a obrigatoriedade do seu nome aparecer em letras maiúsculas, pois as chains são case-sensitive (OUTPUT $\neq$ output). As *chains* criadas pelo utilizador (root) são, pelo contrário, possíveis de apagar e renomear, sendo que a sua principal restrição reside no impossibilidade de atribuir nome igual ao das *chains builtin*.

No kernel as regras e *chains* são guardadas em tabelas utilizando para tal a *framework* Xtables. Estas, ao contrário das *chains* não podem ser criadas, existindo no entanto, por defeito (*builtin*), sendo estas, a *filter*, a *nat* e a *mangle* as quais serão abordadas de seguida:

filter – Das três tabelas referidas é a mais importantes, é utilizada por defeito e contém três *chains builtins*, nomeadamente:

INPUT – Utilizada em todos os pacotes que entram na máquina pelos diferentes interfaces de rede.

OUTPUT – Utilizada em todos os pacotes que saem da máquina por qualquer interface de rede.

FORWARD – Utiliza-se esta chain se os pacotes forem redireccionados para outro qualquer interface de rede ou máquina.

É também importante referir que nas ligações locais (no localhost/na máquina) utiliza-se apenas esta tabela com as chains INPUT e OUTPUT.

nat – Como o próprio nome indica é utilizada aquando do uso de NAT ou suas propriedades, ou seja, em encaminhamento de conexões, SNAT, DNAT, port forwarding, masquerading, proxy transparente ou mesmo em Port Address Translation (PAT ou NAPT). Esta tabela é constituída por 3 *chains builtins*:

PREROUTING – Utilizada quando os pacotes precisam ser modificados logo que entram na máquina. Uma das suas utilizações é a realização de DNAT e redireccionamento de portos.

OUTPUT – Utilizada quando os pacotes locais precisam ser modificados antes de serem encaminhados. Esta chain é apenas consultada para conexões que têm origem em IP's de interfaces locais.

POSTROUTING – Utilizada quando os pacotes precisam de ser modificados após o encaminhamento, principalmente para a realização de SNAT e IP Masquerading.

mangle – Esta tabela é utilizada em situações muito específicas, como por exemplo em caso de alterações nos pacotes, recorrendo para isso a todas as *chains builtins*:

INPUT – Utilizada quando os pacotes necessitam de ser modificados antes de serem enviados para o *chain* INPUT da tabela filter.

FORWARD – Contém as regras para os pacotes que precisam de ser modificados antes de serem enviados para o *chain* FORWARD da tabela filter.

PREROUTING – Nesta *chain* são alterados os pacotes que necessitam de serem processados antes de serem enviados para o *chain* PREROUTING da tabela nat.

POSTROUTING – É utilizada quando os pacotes precisam ser modificados antes de serem enviados para o *chain* POSTROUTING da tabela nat.

OUTPUT – As regras desta *chain* modificam os pacotes antes de eles serem enviados para o *chain* OUTPUT da tabela nat.

3.12.3 Protocolos e Portos

Com o iptables conseguimos ter um controle sobre o tipo de tráfego que pode transitar, sendo possível definir o IP ou mesmo o porto de destino e origem. Para tal são suportados os seguintes protocolos: *User Datagram Protocol* (UDP), TCP e ICMP para os quais é possível definir opções avançadas tornando a flexibilidade do iptables num ponto chave. Este modo de operação é normalmente conhecido por *match* (correspondência). Sendo de seguida expostas as possíveis opções para cada um dos protocolos ou módulos avançados.

Protocolo UDP

- Porto de destino (–dport) – Define o porto de destino de um pacote, para tal é possível atribuir um valor numérico, usar o nome de uma rede de serviços (ex: www ou dns) ou ainda, definir um intervalo de portos a serem abrangidos, sendo que o intervalo deve ser especificado separando os dois números com dois pontos (:). É necessário ter em atenção que quando acrescentado o carácter ponto de exclamação (!) na introdução de uma regra, ocorre uma inversão na correspondência.
- Porto de origem (–sport) – Especifica qual o porto de origem do pacote seguindo as mesma regras/sintaxe que a especificação do porto de destino.

Protocolo TCP

- Portos de destino e origem(–dport/–sport) - Seguem a mesma especificação e regras do protocolo UDP.
- Estados do TCP(–tcp-flags) – Permite estabelecer correspondência a bits de informação específicos ou a estados da ligação TCP.
- Estado da ligação (–state) – Compara o estado de um pacote com os seguintes estados:

  ESTABLISHED – O pacote está dependente de outros pacotes e tem uma conexão estabelecida.

INVALID – Quando o pacote não se encontra ligado a nenhuma conexão activa.

NEW – O pacote é novo, ou seja, está a criar uma nova ligação ou é parte de uma ligação bidireccional desconhecida.

RELATED – O pacote está a criar um nova ligação encontrando-se relacionado com uma ligação activa.

Protocolo ICMP

- Tipo de ICMP (–icmp-type) – Especifica o nome ou número do tipo de ICMP a que queremos fazer correspondência.

Módulo de limite – Explicita um limite máximo de pacotes que a máquina pode receber em determinado período de tempo.

- Limite Total (–limit) Determina o número limite de vezes que um tipo de pacote foi recebido em determinado período de tempo, o qual se for omisso, por defeito será de 3/hora.
- Limite imediato (–limit-burst) – Permite especificar o número máximo de pacotes que se podem receber num curto período de tempo, ex: por minuto. Esta opção deve ser usada em conjunto com a opção –limit.

Módulo MAC – Permite efectuar correspondência a um MAC.

- Efectua a correspondência ao endereço MAC do interface de rede que enviou o pacote. Quando necessário pode ser usado o ponto de exclamação para inverter a correspondência.

3.12.4 Destinos (Targets)

Para cada regra deve ser adicionado um destino, esse destino pode passar pela aceitação ou rejeição do pacote, entre outros, nomeadamente:

DROP : Procede à rejeição do pacote sem notificar a fonte.

ACCEPT: Aceita um pacote e deixa-o passar as regras da firewall.

REJECT: Não aceita o pacote, notificando a fonte da rejeição.

MARK: Permite marcar o pacote, o que posteriormente auxiliará na tomada de decisões ao nível do encaminhamento, bem como na definição da prioridade do tráfego;

LOG: Imprime um log de todos os pacotes que satisfaçam a regra.

ULOG: Guarda um log de todos os pacotes que satisfaçam a regra, utilizando a *framework* de ulog.

DNAT (Destination NAT) : Altera o endereço de destino dos pacotes.

MASQUERADE : Executa IP masquerading, sendo em parte idêntico ao SNAT possuindo, no entanto funcionalidades adicionais.

SNAT (Source NAT) : Altera o endereço de origem dos pacotes.

REDIRECT : Redirecciona os pacotes IP para outro porto da própria máquina.

MIRROR: Troca os endereços de origem e destino.

RETURN : Sai da cadeia de processamento actual, retornando à cadeia anterior.

Tendo em conta as tabelas referidas anteriormente, um dos destinos muito utilizados, por exemplo, para a tabela nat são a Source NAT(SNAT) e Destination NAT(DNAT). O funcionamento do SNAT modifica o endereço de origem do pacote mantendo inalterado o porto de origem. Esse processamento é feito na *chain* POSTROUTING e tem a vantagem de ser mais rápido que o método MASQUERADE, pois não necessita de gerir os IPs e portos disponíveis. No entanto, quando não existe um IP fixo a solução terá obrigatoriamente que passar pelo método MASQUERADE.
Relativamente ao DNAT, esta modifica o endereço de destino traduzindo os IPs externos para o IP do servidor responsável, sendo processada na chain PREROUTING e/ou na OUTPUT. Tal como a SNAT também é mais rápida que o método MASQUERADE, pois também não traduz os portos.

3.12.5 Libiptc

O iptables é constituído por várias bibliotecas entre as quais se destaca a libiptc. Esta é executada em *UserSpace* e permite comunicar com a API do netfilter que reside no kernel. Por seu lado o netfilter é o código interno do kernel responsável pela tarefa de gestão de pacotes e firewall. Assim, para se poder configurar o comportamento da firewall deve utilizar-se a libiptc que nos permite o acesso ao interface do kernel de forma simples e rápida. Podem ainda ser adicionadas outras vantagens à utilização desta biblioteca, por exemplo, o facto de ser bastante madura e ter um API estável.

3.13 Trabalho Relacionado

Esta sub-secção centra-se na abordagem de trabalhos e soluções propostas, por outros grupos para os problemas descritos anteriormente. Para cada trabalho proposto será focado o seu modo de funcionamento, as virtudes e problemas dessas mesmas soluções.

3.13.1 Identificação do servidor de Autenticação

Em [3] é proposta uma alteração ao EAP, que a seguir à display-string define a inserção do termo *NAIRealms* o qual contém uma lista de domínios suportadas pelo *Network Access Server*

(NAS). Permitindo desta forma que o *supplicant* durante a troca de mensagens EAP-Request/Identify verifique se aquele NAS pertence ao domínio a que se deseja autenticar. Um problema desta abordagem é que a estação só tem acesso a estas informações após se ter associado ao AP, traduzindo-se assim, em significativos atrasos no processo de autenticação e descoberta de domínios, necessitando ainda de alterações no *supplicant* e no servidor de autenticação.

A norma IEEE 802.11u em desenvolvimento, introduz um mecanismo em que os AP's (NAS) divulgam uma lista de domínios que servem e os respectivos servidores de autenticação.

3.13.2 Autenticação do servidor de autenticação

O registo DNS CERT [12] quando usado com o DNSSEC permite tornar o DNS numa PKI. Esta proposta suporta diferentes tipos de certificados podendo eles ser, PKI, SPKI, PGP, entre outros. Uma das grandes desvantagens desta proposta é que o tamanho médio de um certificado é geralmente grande e possibilidade de existir um grande número de certificados para o mesmo FQDN, o que tende a carregar substancialmente o DNS.

A outra solução é a descrita pelo grupo *Trans-European Research and Education Networking Association* (TERENA) permite que os diferentes servidores de autenticação comuniquem directamente entre eles sem necessidade de comunicar com um ou mais servidores proxies RADIUS. Para esse efeito é utilizado o DNSSEC que averigua se o domínio desejado ao qual o *supplicant* se quer autenticar pertence ao conjunto de domínios ao qual o AS está autorizado a fazer *roaming*. Esta solução apesar de utilizar o DNSSEC para a verificação do domínio, recorre na mesma ao PKI para verificação dos certificados, logo é sempre necessário ter-se o certificado do AS remoto instalado para verificar a autenticidade dos certificados.

Capítulo 4

Solução proposta

Com vista a resolver os problemas da correcta identificação do servidor de autenticação, bem como da validação do servidor de autenticação, foi criada uma arquitectura que possui como permissas a autenticação e validação mútua do cliente e servidor de autenticação bem como um tipo de construção especial do SSID.

Essa arquitectura permite validar os certificados usados durante a autenticação e no geral possui 2 fases distintas, nas quais é inicialmente efectuada uma autenticação mútua, ocorrendo de seguida uma validação da autenticação. Podem ser identificados na figura 4.1 os blocos básicos que consituem uma autenticação ScalSec.

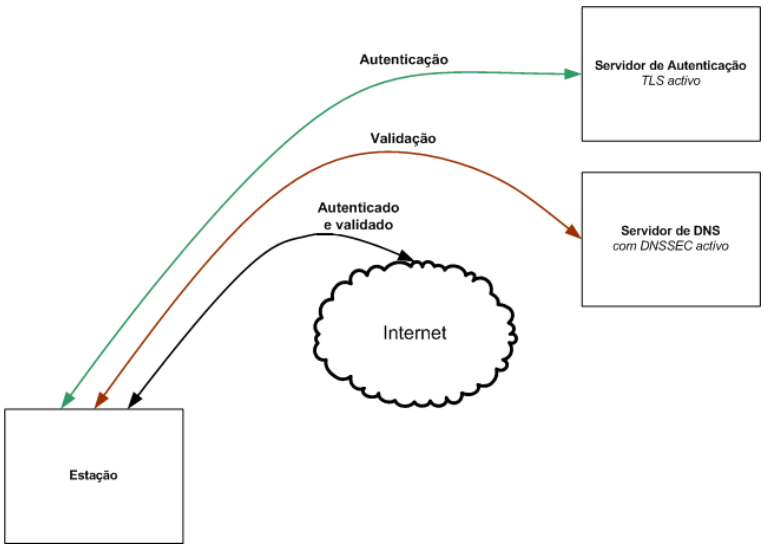


Figura 4.1: Princípios da arquitectura ScalSec

Pretende-se com a introdução desta arquitectura tornar a identificação das rede Wi-Fi única, resolvendo também o problema dos certificados que actualmente são caros e de díficil gestão.

Tendo como meta o alcançar os objectivos traçados, foram criadas e desenvolvidas duas soluções que podem operar em separado uma da outra, mas que quando operam em conjunto complementam-se, são estas, o SSID global e a arquitectura ScalSec.

A solução *SSID global* que se desenvolveu permite tornar a identificação da rede Wi-Fi *única* possibilitando também a correcta identificação do servidor de autenticação.

O modelo actualmente vigente na certificação, o PKIX possui alguns problemas.Devido à sua arquitectura e premissas, a *framework* é rígida e não muito viável em certos cenários de uso, como no caso de single sign-on ou na gestão das chaves públicas. Veja-se por exemplo o modo de revogação de uma chave que é um método complexo, de difícil execução e que traz custos ao cliente, sendo que a arquitectura ScalSec possui outras premissas e procedimentos que tornam o processo de validação mais simples, eficiente, seguro e versátil.

A arquitectura ScalSec foi pensada pelo co-orientador desta dissertação o Eng. Jaime Dias, ele é o grande mentor desta nova arquitectura que visa aproveitar ao máximo as funcionalidades do DNSSEC nas mais diversas vertentes. Aqui é apresentada uma parte da arquitectura ScalSec, sendo que neste caso específico é utilizada para autenticação de redes Wi-Fi possuindo esta um modelo de abrangência maior. Tem na sua base os mesmos princípios do DNSSEC e introduz um novo conceito de certificação. Para tal, a validação dos certificados é feita através da utilização de novos registos DNSSEC.

Desta forma a arquitectura ScalSec consegue tornar o servidor de DNS local numa PKI global, sem que os utilizadores deste serviço necessitem ter o certificado de CA localmente instalado no cliente. Os novos registos permitem ainda ter um âmbito de acção mais alargado e dotar a arquitectura de um espectro de acção mais amplo.

A certificação ScalSec resolve o problema dos certificados serem caros e de difícil gestão, pois com esta certificação é possível utilizar certificados *self sign* (certificados que se assinam a si próprios) de baixo custo.

Na figura 4.2 é possível identificar o cenário de autenticação em que se utilizam as duas soluções em simultâneo, tanto a solução do *SSID global* como a solução ScalSec.

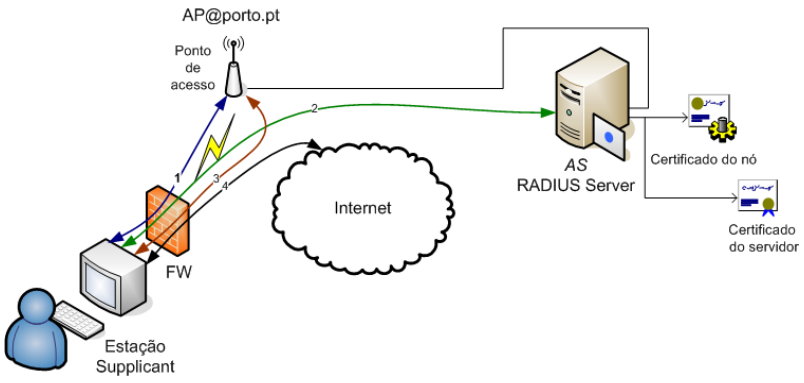


Figura 4.2: Processo de autenticação que utiliza a arquitectura desenvolvida

Uma das permissas da arquitectura ScalSec é identificar o FQDN de cada organização como sendo um nó. Cada nó pode conter diversos serviços possuindo apenas um certificado de raiz de CA, o qual é responsável pela assinatura dos certificados dos serviços aí disponíveis, chamado de certificado do nó.

Em paralelo à adopção da arquitectura ScalSec é possível providenciar o serviço de DNSSEC à rede infra-estruturada da organização.

4.1 SSID global

Após na secção 3.6 ter sido feita uma descrição da forma como as redes Wi-Fi são identificadas, foi constatado que existe um problema na identificação das redes Wi-Fi (ESS).

Normalmente esse problema pode ser evitado através de um planeamento correcto da rede Wi-Fi. Verificando-se a existência de um SSID igual ao que se pretende nas redes vizinhas, essa regra diminui os problemas que possam vir a surgir. Mas no entanto não impede que uma pessoa mal intensionada crie uma rede com o mesmo SSID.

A solução encontrada para a correcta identificação do servidor de autenticação e criação de um SSID único ocorreu com a introdução de uma nova semântica no SSID.

Com a introdução do *SSID global*, o SSID passa agora a conter um caracter especial, neste caso o '@', que delimita logicamente o SSID (normal) de um FQDN o qual corresponde ao nome no domínio do servidor de autenticação. Desta forma, consegue-se associar o SSID de uma rede ESS ao seu servidor interno de autenticação. O SSID ao conter a identificação do servidor de autenticação através do FQDN passa a ser único e por isso a denominação de global.

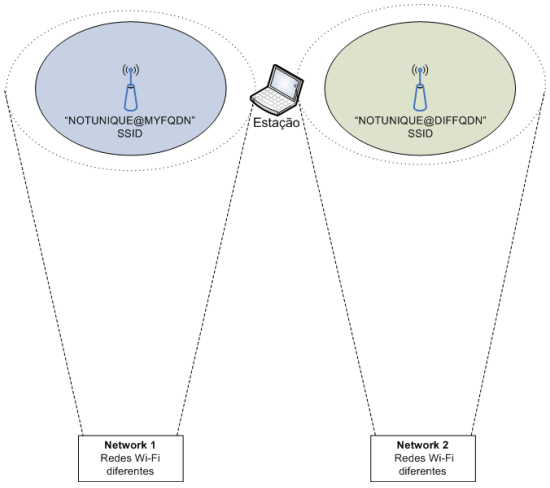


Figura 4.3: Identificação da solução 1/1 descrita na secção 4.1

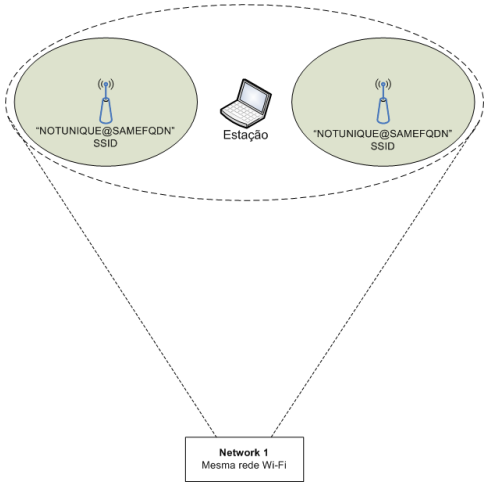


Figura 4.4: Identificação da solução 2/2 descrita na secção 4.1

No entanto, de modo a melhorar a verificação do servidor de autenticação adicionou-se uma regra suplementar na verificação dos certificados. Tendo em conta o processo de autenticação que utiliza o PEAP e por consequência certificados X.509, torna-se obrigatório que o campo Alternative Subject Name, especifique o FQDN que está atribuído ao NAS. Desta forma, associam-se formas de identificação alternativas ao titular do certificado.

Este campo permite especificar cinco tipos de identificação URI. Aquele que será aqui utilizado é do tipo DNS (podendo ser outro). Sendo assim o *Alternative Subject Name* será do género: DNS:*«FQDN»*. Foi assim feito para manter compatibilidade com as regras estabelecidas pela PKI.

Desta modo é possível identificar o servidor de autenticação pretendido de forma correcta, baseando-se apenas na informação que o utilizador costuma inserir/seleccionar do SSID. Esta solução pode ser utilizada em separado da solução ScalSec.

Parte de um certificado que segue estas regras:

```
X509v3 Subject Key Identifier:
       XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX
X509v3 Authority Key Identifier:
       keyid:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX
X509v3 Subject Alternative Name:
       DNS:radius.porto.pt
```

Em suma, esta solução mantém a compatibilidade com o formato anterior que não possuía qualquer regra, não requerendo qualquer alteração ao protocolo 802.11, nem ao AP ou servidor de autenticação. O único software que necessita de ser alterado é do lado o cliente, sendo necessário alterar o *supplicant* para que este reconheça a nova semântica e lhe atribua o seu devido significado.

Uma falha que pode ser apontada a esta solução é o facto do FQDN poder ter até 255 bytes, mas o SSID ser limitado a 32 bytes. A acrescentar a este facto é utilizado obrigatoriamente um caracter separador '@' restingindo assim o FQDN a um máximo de 31 bytes (caractéres) o que pode em certos casos ser demasiado restritivo.

Com a introdução desta regra no processo de criação do SSID consegue-se suplantar o problema da identificação do servidor de autenticação, mantendo-se no entanto o problema da validade do mesmo.

4.2 ScalSec

De modo a tornar a identificação mais fácil relativamente às contribuições efectuadas, bem como para uma melhor compreensão desta arquitectura e seus requisitos, serão descritos os blocos funcionais identificados na introdução deste capítulo. Os blocos que não necessitam de trabalho específico tem uma cor mais escura que os blocos que necessitam de ser implementados ou modificados.

Começando pela análise do bloco DNS que tem como objectivo providenciar o serviço de DNS e a inclusão dos registo ScalSec, este é constituído pelo servidor de DNS mais a *framework* de gestão dos registos.

O outro bloco funcional é o do serviço de autenticação (RADIUS) que conterá o servidor de RADIUS, os certificados e a *framework* de certificação e gestão.

Note-se que as alterações necessárias nos blocos dos serviços são reduzidas sendo apenas necessário com a introdução da arquitectura ScalSec novos blocos lógicos que garantam a facilidade de gestão e manutenção desta solução.

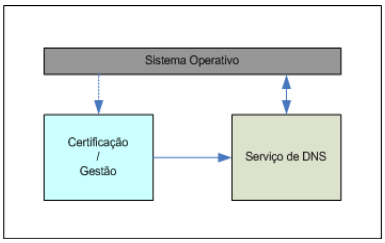


Figura 4.5: Bloco do serviço de DNS

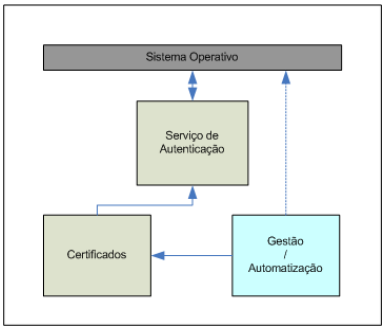


Figura 4.6: Bloco do serviço de RADIUS

O bloco funcional estação é o que necessita de um número maior de alterações e consequentes adições de blocos lógicos. Este bloco possuirá o *supplicant*, o lwresd, o libiptc, o validador ScalSec e o identificador de *SSID global*.

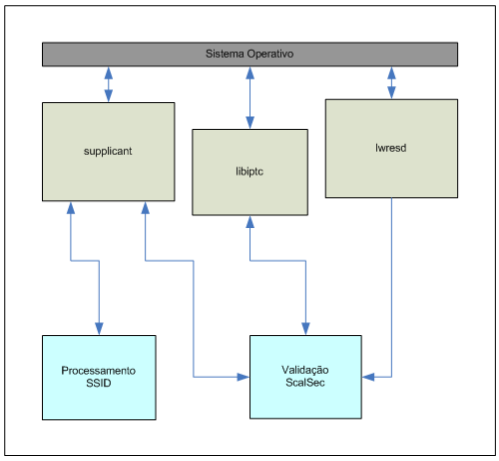


Figura 4.7: Bloco da arquitectura do cliente

A figura 4.8 mostra com um pouco mais de pormenor os blocos que actuam ao longo desta arquitectura, identificando também as diferentes etapas da mesma.

Também se identificam os passos necessários para que uma autenticação ScalSec ocorra, no entanto todo o processo de autenticação e certificação será abordado com mais detalhe nas próximas sub-secções.

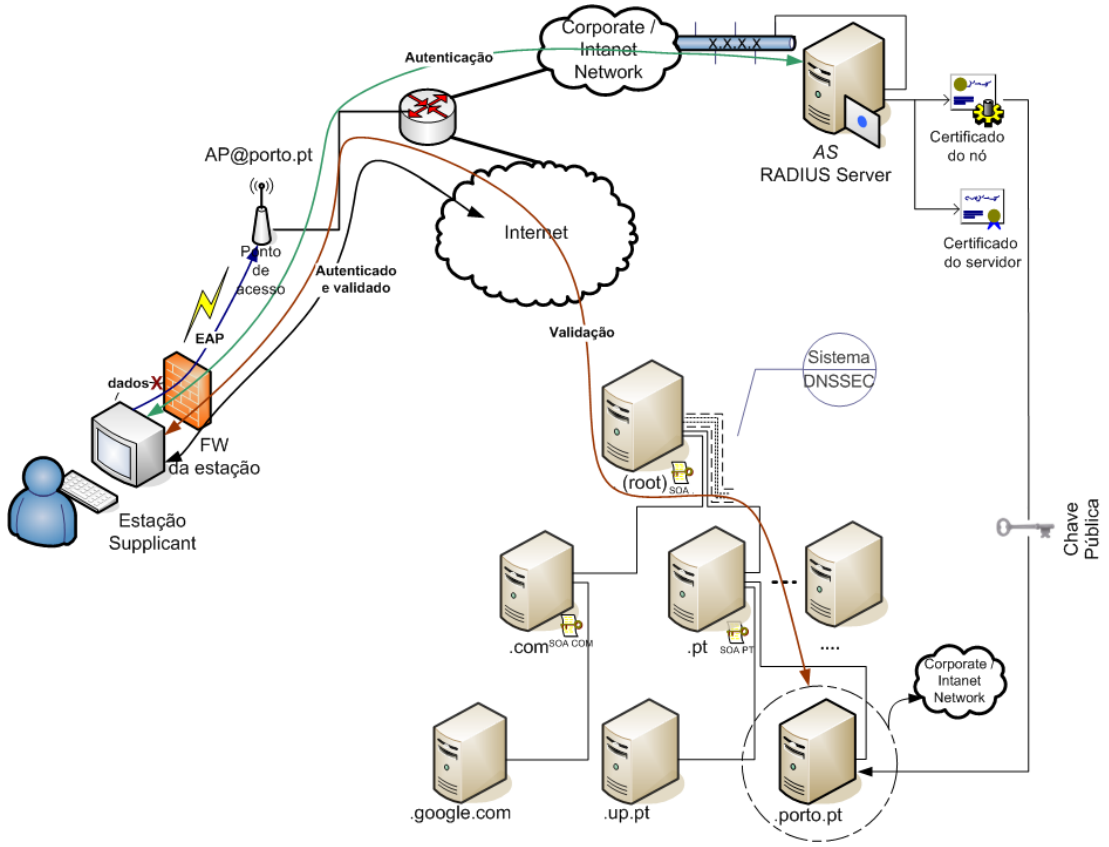


Figura 4.8: Arquitectura implementada

4.2.1 Autenticação do servidor de autenticação

O modelo ScalSec permite autenticar mutuamente o servidor de autenticação e utilizador só deixando passar tráfego após ambos terem sido correctamente autenticados. Esta arquitectura não necessita de possuir o certificado do nó para executar a cadeia de certificação, ao contrário do que acontece no modelo PKIX.

Para uma melhor descrição do processo de autenticação deve-se consultar o anexo 7.2 no qual se dá um maior detalhe acerca do processo de autenticação.

De seguida será explicado o procedimentos de certificação ScalSec aquando de uma autenticação com um servidor de autenticação que tenha TLS. Os procedimentos divergem um pouco dos modelo PKIX. Assim no momento da criação do túnel TLS é feita a verificação das regras dos certificados, com a excepção da verificação ao certificado de CA que só é parcialmente feita, pois apenas se verifica se é um certificado válido e se contém as extensões obrigatórias. Saliente-se que não é feita toda a cadeia de certificação *(DPV)* uma vez que não é possível comparar a chave pública do certificado de CA com uma chave pública conhecida.

De modo a permitir uma maior segurança na solução apresentada e deste modo impedir que a estação se autentique em redes forjadas, foram introduzidas um conjunto de regras nas diferentes fases de autenticação. De um modo geral pode-se repartir a certificação em três fases distintas. Na primeira fase é feita uma pré-verificação dos certificados envolvidos na autenticação, numa

segunda fase é obtido um conjunto de registos do DNSSEC que possam comprovar a autenticidade do certificado e na terceira e ultima fase é feita a comparação do resumo da chave pública do certificado do nó com os registos obtidos.Se a comparação for bem sucessida então estaremos perante um certificado de nó válido. Assim, para a primeira fase, as regras que foram definidas são as seguintes: o certificado de CA deve ter explicitado no *Common Name (CN)* o valor ScalSec, por seu lado o certificado do AS deverá ter o *CN* do Issuer com o valor ScalSec. Para as extensões dos certificados são, no caso do certificado do nó, a obrigatoriedade de conter basicConstrains o valor CA=TRUE, por sua vez no certificado do servidor é necessária a presença da extensão Alternative Subject Name=<tipo>:radius.

Numa segunda fase dever-se-ão obter os registos DNSSEC do FQDN anunciado no *SSID global* quer seja através de um ficheiro instalado localmente, quer seja através da Internet usando o DNSSEC (método preferencial), os registos obtidos serão do tipo H para o FQDN. Na última fase é feita uma comparação do resumo da chave pública do certificado de CA, que foi utilizado na negociação TLS, com o resumo do registo correspondente obtido. Se os valores forem iguais verificámos a validade do certificado de CA e por consequência autentica-se no NAS, considerando-o fidedigno.

Se tudo correr bem, entrar-se-á na segunda fase de autenticação ScalSec no qual o *supplicant* já negociou o 4 way handshake e é lançada um processo que será responsável pela obtenção e verificação das chaves, para que possa passar para o estado final de autenticado. Nesta fase o acesso a registos de DNS já é possível pois a estação já está 'pré-autenticada', o *supplicant* obtém os registos do DNS a que se referem o FQDN anexado no *SSID global* a que a estação se ligou e é feita uma validação sobre os registos de DNS e sobre as credenciais apresentadas pela NAS. Enquanto essa validação não for feita todo o tráfego da estação é bloqueado, sendo que se a validação for positiva a estação poderá aceder normalmente à Internet, se algum erro ocorrer a estação abortará a ligação àquele AP.

Procedeu-se ainda a uma alteração no modo de validação da redes Wi-Fi que não permite o uso de protocolos de autenticação sem cifra, por exemplo PAP aquando do uso do *SSID global*.

Para a validação dos registos é usado o DNSSEC, segue o esquema da figura 4.8.

4.2.2 Certificação

O processo de certificação no modelo ScalSec pode ser separado em duas partes distintas, a certificação dos servidores e a dos registos e certificados.

Para a primeira, certificação das aplicações (serviços), esta é feita pelo sistema operativo e suas restrições de acesso recorrendo também à *framework* de gestão para a automatização de todo esse processo.

Para a certificação dos certificados e registos é usado o conceito de certificado de nó que funciona como a CA para todos os serviços desse nó. Ao utilizar-se o registo H, estabelece-se a associação que falta entre a chave pública do certificado usado na autenticação TLS e o nome associado ao FQDN ou IP anunciado no SSID.

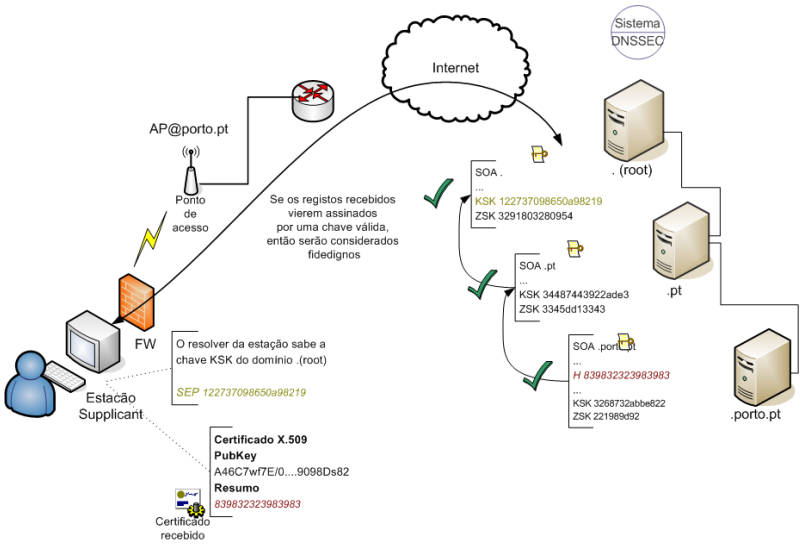


Figura 4.9: Validação dos registos ScalSec

A certificação do modelo ScalSec assenta no DNSSEC, garantindo-se assim que os registos H são fidedignos até ao SEP sendo este necessário de configurar no *resolver* do cliente.

Para a criação dos registos H é necessário ter uma referência do certificado do nó, sendo que a referência escolhida foi o resumo da chave púbica do certificado.

A actualização dos registos H é feita de um modo automático pela *framework* de gestão desenvolvida e segue os passos identificados na figura 4.10.

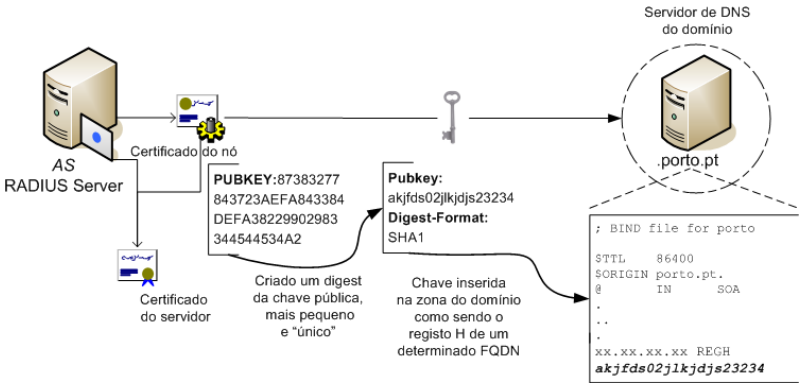


Figura 4.10: Actualização dos registos H de DNS

4.2.3 Registos ScalSec

Esta arquitectura para a verificação e validação das credenciais do utilizador recorre a dois registos de DNS específicos, o registo de tipo K e o registo de tipo H. Vai-se abordar quais os objectivos que este registos têm dentro desta arquitectura.

4.2.3.1 Registo K

Este novo registo é baseado no registo DNSKEY, sendo no entanto mais simples que este e por estar adaptado a uma utilização específica, é mais eficiente. Permite guardar chaves de hosts ou utilizadores para aplicações como por exemplo gnupgp, ssh ou EAP (no caso de obtenção de chaves dos utilizadores). Este registo não será objecto de mais aprofundamento pois os objectivos do mesmo não se enquadram no tema desta dissertação, no entanto acha-se pertinente dar uma perpectiva global da arquitectura implementada.

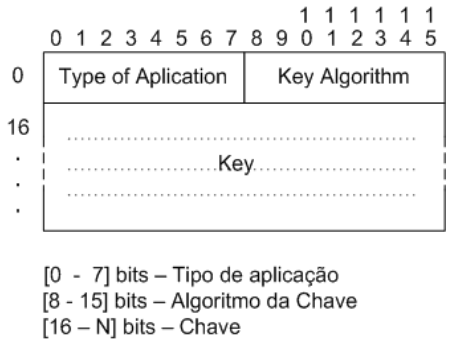


Figura 4.11: Formato do Registo K

4.2.3.2 Registo H

O registo H é um RR usado para obter hashs de chaves, baseado na estrutura do registo DS. Para este registo decidiu-se defini-lo como sendo o registo do tipo 120, é um valores possíveis de definir pela tabela da IANA. Este registo foi desenvolvido por forma a suportar os últimos algoritmos de hash cripográficos seguros suportando por isso o SHA-1 e o SHA-2, DH e EC. Existe no entanto a possibilidade de com facilidade acrescentar outros algoritmos de resumo a este registo. Tendo como vantagem suportar diferentes tipos de resumos (hashs). Para uma maior eficiência, o tamanho do registo é dinâmico, variando em função do tipo de resumo e número de bits a este inerentes.

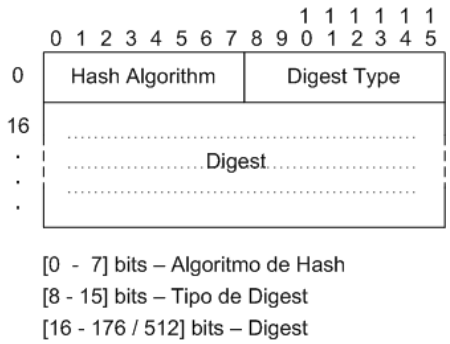


Figura 4.12: Formato do Registo H

A tabela 4.1 associa os diferentes algoritmos de resumo SHA possíveis de utilizar, ao tamanho do resumo (número de bits).

Algorimo	Tamanho Final (bits)	Número de Internos (bits)	Tamanho do Bloco (Encriptação)
SHA-1	160	160	512
SHA-256/224	256/224	256	512
SHA-512/384	512/384	512	1024

Tabela 4.1: Tamanho total dos diferentes algoritmos de SHA

A tabela 4.2 refere os diferentes tipos de chaves idealizados que deviam estar dísponíveis e suportadas.

Algoritmos				
RSA/SHA1	DSA/SHA1	RSA/SHA2	DSA/SHA2	ECC

Tabela 4.2: Tipo de algoritmos de chaves idealizados para o registo H

A representação do registo H numa zona deverá ter o formato descrito no figura 4.13. É possível verificar a constituição do registo H, este é constituído pelo algoritmo da chave do resumo, pelo algoritmo do resumo e pelo resumo. O algoritmo da chave é utilizado para se poder verificar a chave de um certificado. No entanto, a chave pública de um certificado pode ser derivada de vários tipos de algoritmos, daí a necessidade deste campo. O outro campo que constitui o registo é o algoritmo do resumo que permite dar uma maior flexibilidade a esta arquitectura pois ao permitir a escolha de algoritmos de resumo, dá-se a possibilidade de adaptar o melhor algoritmo de resumo ao uso que se pretende ter deste registo.

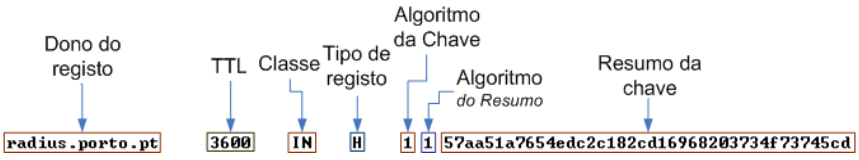


Figura 4.13: Possível formato registo H nas zonas de DNS

4.3 Conclusões

Com a introdução desta arquitectura o modelo de certificação dispõe de um sistema flexível, distribuído e com custos reduzidos.

A arquitectura que foi definida atráves do uso do *SSID global* permite resolver o problemas que a falta de regras na definição do SSID provoca, tendo sido definida uma semântica que permite a correcta identificação das rede Wi-Fi, bem como do servidor de autenticação.

Foram descritos os diversos blocos funcionais que interagem neste tipo de arquitectura e explicados os diferentes procedimentos necessários para a aplicação deste modelo.

Com o modelo ScalSec introduz-se o conceito de resumo de uma chave por serviço por nó, tendo cada nó por sua vez um certificado que assina todos os serviços aí dísponíveis. Esta arquitectura é orientada para um espaço de nomes ao nível global sendo a confiança dos diversos servidores até ao nó estabelecida pelas entidades que gerem o espaço de DNS do nó. Foram também definidos novos registos DNSSEC que permitem que o processo de certificação do modelo ScalSec seja um modelo seguro e eficiente.

Capítulo 5

Implementação

Neste capítulo é focado o modo de implementação. Para uma melhor compreensão deste capítulo, procedeu-se à separação do mesmo, de acordo com as fases lógicas do processo de autenticação, bem como as soluções a apresentar.

5.1 Configuração do servidor de AAA

O protocolo AAA usado escolhido foi o RADIUS. Para que o servidor tenha as funcionalidades pretendidas é necessário reunir-se os seguintes requisitos, os certificados e o servidor de autenticação. Nas próximas sub-secções são descritos os métodos e ferramentas utilizados para cumprir cada um dos requisitos acima descritos.

5.1.1 Geração de Certificados

Nesta sub-secção são explicados os procedimentos necessários para gerar os certificados do servidor de autenticação. Nestes, o campo *Alternative Subject Name* tem de ser o FQDN anunciado no *SSID global*. Assim, é necessário instalar o openssl, programa utilizado para a geração dos certificados.

Decidiu-se utilizar um script que permite a geração automática dos certificados, o gen_certs, sendo estes o certificado do nó ou o certificado do servidor de autenticação. Para tornar o processo automático é necessário gerar 2 ficheiros de configuração distintos, um para o certificado do nó outro para o certificado do servidor. Os ficheiros de configuração bem como o script podem ser consultado nos anexos.

O processo de geração de certificados pode ser definido em cinco passos:

- Inicialmente é gerada um chave privada com um determinado número de bits, que é a chave privada "superior".
- É de seguida gerado um pedido para um certificado *(CSR)*, sendo para isso necessário providenciar um conjunto de dados de identificação do certificado, bem como, a password que estará associada ao certificado.

- O terceiro passo passa por pegar no pedido para um certificado *(CSR)*, na chave privada gerada e dar início ao processo de auto-assinatura *(self sign)*, desta forma temos um certificado auto-assinado que pode ser usado como um certificado do nó.
- Falta agora gerar a chave privada e o pedido do certificado do servidor, o processo é semelhante ao do certificado do nó, sendo que é gerada uma chave e o pedido de certificado não assinado.
- Por último o certificado do servidor é assinado pelo certificado do nó, anteriormente gerado.

Os ficheiros de configuração que se utilizou para a geração de certificados, são constituídos por diferentes secções, sendo que cada secção tem um ou mais atributos configurados. Cada secção é delimitada por '[<nome de secção>]' e cada atributo utiliza uma notação do género <atributo>_default=<valor>, repare-se no nome do campo que vem sempre referenciado como <nome do atributo>_default, desta forma consegue-se tornar o processo de geração automático, sem necessidade de serem feitas perguntas ao utilizador para o preenchimento do certificado.

Enxerto de um ficheiro de configuração:

```
[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = PT
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Porto

localityName = Locality Name (eg, city)
localityName_default = Porto

0.organizationName = Organization Name (eg, company)
0.organizationName_default = INESC

# we can do this but it is not needed normally :-)
#1.organizationName = Second Organization Name (eg, company)
#1.organizationName_default = World Wide Web Pty Ltd

organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = INESC Porto

commonName = Common Name (eg, YOUR name)
commonName_default = ScalSec
commonName_max = 64

emailAddress = Email Address
```

```
emailAddress_default = hostmaster@inescp.pt
emailAddress_max = 64
```

.....

```
# Include email address in subject alt name: another PKIX recommendation
subjectAltName=DNS:radius.porto.pt
# Copy issuer details
# issuerAltName=issuer:copy
```

Existe um modo de adicionar as extensões necessárias automaticamente ao nosso certificado. Para isso, no ficheiro de configuração do certificado, o local do certificado do nó e a sua chave privada tem de estar correcto.

```
 [ CA_default ]

dir = ./CA # Where everything is kept
certs = $dir/certs # Where the issued certs are kept
crl_dir = $dir/crl # Where the issued crl are kept
database = $dir/index.txt # database index file.
#unique_subject = no # Set to 'no' to allow creation of
# several certificates with same subject.
new_certs_dir = $dir/newcerts # default place for new certs.

certificate = $dir/cacert.pem  # The CA certificate
serial = $dir/serial  # The current serial number

private_key = $dir/private/cacert.key # The private key
```

Sendo ainda necessário, configurar as extensões que serão adicionadas ao certificado do servidor quando é assinado por uma CA. Isso é feito através da configuração do seguinte atributo.

```
x509_extensions = v3_ca # The extentions to add to the self signed cert
```

Gerados os certificados, segue-se a configuração do servidor de autenticação, mais propriamente um servidor RADIUS.

5.1.2 Configuração do freeRADIUS

Antes de se dar início o processo de configuração do servidor de autenticação, procedeu-se à instalação do freeRADIUS, uma implementação open source do protocolo RADIUS. Como é usado um modo de autenticação que usa o TLS é necessário criar o pacote do freeRADIUS à mão pois o módulo EAP-PEAP depende do openssl que não possui uma licença compatível com o GPL. Dessa forma, foi necessário o download do código fonte do freerRADIUS, bem como a instalação das dependências que o programa necessitava para compilar. Após a criação do pacote do freeRADIUS e a sua instalação é necessário configurá-lo.

A configuração do servidor RADIUS, não é um processo muito complexo, pois, apesar de envolver a configuração de diversos módulos, cada um destes módulos fica restrito a um ficheiro, facilitando assim a tarefa.

Passa-se de seguida a enunciar as alterações que foram necessárias fazer aos diferentes ficheiros de configuração.

No ficheiro 'radiusd.conf' é possível configurar os diferentes módulos operacionais do RADIUS. Sendo necessário configurar o módulo de autorização, para que habilite o suporte 'eap'

```
 authorize {
        #
....
        #
        #  This module takes care of EAP-MD5, EAP-TLS, and EAP-LEAP
        #  authentication.
        #
        #  It also sets the EAP-Type attribute in the request
        #  attribute list to the EAP type from the packet.
        eap
        #
```

É também necessário alterar o módulo de autenticação para que suporte o modo eap

```
 authenticate {
        #
....
        #
        #  Allow EAP authentication.
        eap
....
 }
```

Por requisitos operacionais e como medida de segurança é também preciso configurar o nome do utilizador que executará o servidor, neste caso, o utilizador é o freerad

```
#
user = freerad
group = freerad
```

De seguida, procede-se à identificação das alterações necessárias para o funcionamento do módulo eap, implicando isso, alterações no ficheiro 'eap.conf'. Altera-se o modo predifinido de autenticação o EAP para o *Protected Extensible Authentication Protocol* (PEAP).

```
        eap {
                #  Invoke the default supported EAP type when
                #  EAP-Identity response is received.
.....
                #
                default_eap_type = peap
```

É necessário indicar o local correcto onde se encontram os certificados

```
        tls {
                private_key_password = testing
                private_key_file = ${raddbdir}/certs/servcert.key
.....
                #  name.
                certificate_file = ${raddbdir}/certs/servcert.pem

                #  Trusted Root CA list
                CA_file = ${raddbdir}/certs/CA/cacert.pem
                CA_path= ${raddbdir}/certs/CA
.....
            }
```

É também necessário configurar o PEAP para usar o MS-CHAPv2 como autenticação predefinida.

```
        peap {
.....
                 default_eap_type = mschapv2
.....
                 use_tunneled_reply = yes
             }
```

Findada a configuração do módulo eap, é necessário configurar o módulo responsável pela autorização de acesso ao servidor, que se encontra no ficheiro 'clients.conf'.

```
# CertArt AP Server

client 172.16.10.253 {
       secret    = testing
       shortname = Certart-AP
}
```

Com esta alteração autoriza-se o AP a ligar-se ao servidor de autenticação

Por último, é necessário criar um utilizador para que possamos autenticá-lo. Essa alteração deve ser feita no ficheiro 'users'

```
"testing" Cleartext-Password := "password"
          Reply-Message = "Your account is enabled."
```

Depois de todas estas alterações, nos diferentes ficheiros de configuração devemos iniciar o servidor, em modo debug para verificar se não houve nenhum erro, isso pode ser feito executando o seguinte comando "radiusd -X". Existe também, um outro comando que nos permite saber se existe algum erro nas alterações introduzidas nos ficheiros de configuração, o check-radiusd-config.

5.1.3 Scripts de Gestão

Os scripts de gestão foram criados em linguagem BASH e utilizam utilitários da shell linux standard (coreutils). Como nota adicional, é também necessário ter o serviço crontab a correr pois a execução periódica dos scripts depende deste serviço. Os scripts fazem validações do calendário tendo verificações para os dias do mês a que são executadas as actualizações bem como as horas a que é iniciado o processo. Também permite de forma automática utilizar o comando sudo na execução das tarefas, pois a maioria das accções necessita de permissões root.

- **radiusupdate**, é o script principal de gestão que permite a instalação dos diversos scripts e permite de forma ordenada a execução correcta dos outros scripts. O intervalo de tempo com que este script corre depende do crontab, após a instalação do ficheiro de zona é inserida uma entrada no crontab que utiliza o intervalo de tempo definido no cabeçalho do ficheiro.

```
Ajuda:

radiusupdate <dns zone> <options>

Specifies to wich dns zone the radius server is depending on
It criates rules and tasks for a first time use
mirror - it sends a copy
Other variable options:
 $DEBUG - Enable Debug!
```

- **certupdater**, este script é responsável pela execução de diversas tarefas, entre elas, a criação de uma nova pasta para a geração de novos certificados, criação de novos certificados, geração de ficheiros com os registos H para a inclusão nas zonas e apagar os certificados já obsoletos.

```
Ajuda:

certupdater <cert overlay> <options> <status>

Cert overlay - if it's the old | new overlay
options - options to be applied to that key:
          create | del | set
status - status options for each option given:
          default | old | new | dnskey | all | server
Other variable options:
 $DEBUG - Enable Debug!
```

- **put_dnskey**, permite colocar o ficheiro com o registo do tipo H na pasta que contém a zona de DNS passada no primeiro parâmetro, o ficheiro pode ser colocado localmente ou remotamente via ssh (scp), neste último caso, o host de destino pode ser configurado manualmente editando o script e alterando a variável RHOST.

```
Ajuda:

put_dnskey <zone name> -remote

remote - Places the key file in a remote host
Other variable options:
 $DEBUG - Enable Debug!

Please note that you need to have a 'regular' directory map
```

- **radiusupdate-monthly**, script utilizado para fazer a rotação dos certificados do servidor de RADIUS com base nas tarefas mensais, por exemplo, na criação de novos certificados de nó. As tarefas foram separadas em três fases distintas, para mais informações sobre as acções executadas em cada fase consulte a secção 5.2.4. Este script também actualiza as entradas do crontab para a execução da próxima actualização.

```
Ajuda:

radiusupdate-monthly <task>

task- Can be any of this values, 0,1,2, depending on what was the last
executed task.
Please especify what task should be performed
```

- **generate_certs**, este script permite gerar certificados para serem usados pelo freeRADIUS. Os certificados gerados podem ser os certificados de nó, ou novo certificado do nó ou gerar novos certificados de CA e de nó.

```
Please especify what certs want to generate
Ajuda:
generate_certs  <options>
all - CA cert and server cert
ca - Just the CA cert
server - Just the server cert
Other variable options:
 $DEBUG - Enable Debug!
```

5.2 DNSSEC

5.2.1 Configuração do DNS

Para a implementação da solução proposta é necessária a instalação de um conjunto de servidores de DNS, os quais devem possuir a extensão DNSSEC activa.

Para tal, foi instalado o pacote do *Berkeley Internet Name Domain* (BIND) em todas as máquinas utilizadas para simular o serviço de DNS.

Recorreu-se a uma solução com dois níveis de hierarquia, o topo da hierarquia é o nosso domínio .pt, que é montado na Máquina 07, sendo os outros dois sub-domínios, o porto.pt e .inescp.pt montados nas máquinas 06 e 05 respectivamente.

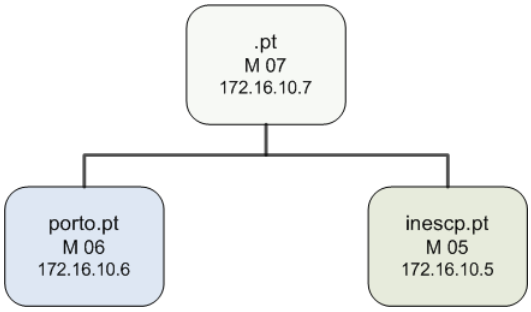


Figura 5.1: Diagrama da arquitectura definida para o DNS

Ambos os servidores dos sub-domínios têm como forwarder o domínio .pt, neste caso o servidor com o endereço 172.16.10.7. Por seu lado, o domínio .pt não tem forwarder para nenhum outro servidor DNS. Assim sendo, apenas estarão disponíveis as resoluções de nome nos sub-domínios, porto.pt e .inescp.pt.

A utilização de DNSSEC não é prejudicial ao normal funcionamento dos servidores de DNS, pois o pedido de resolução apenas tem as extensões DNSSEC se explicitado.

A utilização de dois servidores de DNS de níveis hierárquicos diferentes é necessária, pois para que um pedido de resolução DNSSEC tenha a flag de dados autenticada é necessária uma cadeia não inferior a dois resolvers DNSSEC *(Island of trust)*. Esta é uma regra estabelecida em [15], que explicita a introdução de um *Secure Entry Point* (SEP), o qual pode utilizar um registo *Delegation Signer* (DS), ou a chave pública utilizada para a assinatura da zona, podendo esta ser hierarquicamente inferior ao SEP, ou estar definida no resolver da estação.

Explicar-se-à de seguida em traços gerais o modo de configuração dos servidores DNS dos sub-domínios. Assim inicia-se a explicação com o ficheiro de configuração named.conf o qual permite a especificação de um forwarder para a de resolução de endereços caso se deseje delegar essa tarefa para outro servidor. Assim como é possível especificar qual as interfaces/IP que o servidor deve escutar.

```
options {
        directory "/var/cache/bind";
        dnssec-enable yes;

....
        forwarders { 172.16.10.7; };

        auth-nxdomain no;    # conform to RFC1035

        listen-on { 172.16.10.5; };
};
```

Para se definir uma zona é necessário explicitar o nome da zona, o seu tipo e o ficheiro da zona, não sendo qualquer outro parâmetro necessário.

```
zone "10.16.172.in-addr.arpa" {
       type master;  // tipo, pode ser master ou slave
       file "/etc/bind/10.16.172.in-addr.arpa/db.10.16.172.in-addr.arpa.signed";

       update-policy {
        grant w3.porto.pt. name 6.10.16.172.in-addr.arpa. ANY;
        grant radius.porto.pt. name 8.10.16.172.in-addr.arpa. ANY;
      };

};

zone "porto.pt" {
       type master;
       file "/etc/bind/porto.pt/db.porto.pt.signed";
       // O ficheiro que define a nova zona, constate-se o .signed, querendo isso s
       que se trata de uma zona assinada

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

};
```

As opcções de update-policy são opcionais e destinam-se a possibilitar a actualização das zonas remotamente e de forma dinâmica, recorrendo ao DDNS.

As chaves referidas “w3.porto.pt” e “radius.porto.pt” encontram-se num ficheiro à parte como mandam as boas práticas de gestão e segurança.

Iremos agora analisar um ficheiro de zona, neste caso, a zona porto.pt.

```
$TTL    86400
$ORIGIN porto.pt.
@       IN      SOA     ns01.porto.pt. root.porto.pt. (
                             3         ; Serial
                         86400         ; Refresh
                         43200         ; retentar de 15 em 15 segundos
                        129600         ; Expira, daqui a uma semana
                         86400 )       ; Tempo minimo de duração 3 horas
                                       ; Negative Cache TTL
@        IN      NS      ns01
         IN      MX      10 w3.porto.pt.

; the in-zone name server(s) have an A record
```

```
ns01    IN      A       172.16.10.6  ; Endereço do servidor de DNS do porto.pt
ns01 IN AAAA fe80::250:baff:fec7:f6a8
www IN CNAME w3
w3   IN A 172.16.10.6
w3 IN AAAA fe80::250:baff:fec7:f6a8
radius  IN A 172.16.10.8 ; Endereço onde se localiza o AS do porto.pt

; Cisco Controller
cisco    IN      A       172.16.10.253
cisco  IN      AAAA    fe80::202:a5ff:fed4:21be
```

O passo seguinte é a assinatura da zona, para isso é necessário, primeiro, gerar no mínimo uma chave (ZSK) para assinar a zona. De forma a melhorar todo o processo de gestão de chaves é criado o script gen_key, sendo este usado para a geração de chaves. Este script utiliza o programa dnsssec-keygen, que é instalado juntamente com o pacote do BIND, mas possui um conjunto de comandos mais apelativos e simples.

Além da chave de ZSK, decidiu-se também ter os servidores de DNS com a chave KSK, sendo ela a referência ao longo da cadeia de DNS, uma vez que possui uma validade superior à chave ZSK. Para isso, é mais uma vez usado o gen_key.

Quando usado o ScalSec é necessário publicar os novos registos, sendo preciso obter o ficheiro com o resumo da chave pública do certificado do nó. Para isso, utilizar-se-ão os script put_dnskey e nsupdate, que tornam o processo automático e transparente para o administrador da rede. O primeiro obtém o resumo da chave púlica do certificado do nó e cria o ficheiro hradiusfile.ds e coloca-o num local predefinido, o outro, executa as alterações necessárias à zona de DNS, neste caso, porto.pt.

Assim, o rodapé do ficheiro de zona fica com o seguinte formato:

```
; DNSSEC Keys
; H/K Radius Key
$INCLUDE "hradiusfile.ds"
; KSK Key
$INCLUDE "ksk/Kporto.pt.+005+15852.key"
; ZSK Key
$INCLUDE "zsk/Kporto.pt.+005+50744.key"
```

Para terminar a assinatura de zonas, corre-se o script sign_zone que assina a zona porto.pt, gerando 3 ficheiros, o dssset, o keyset e o ficheiro de zona assinado (.signed). O dsset contém o resumo das chaves utilizadas para assinar a zona, deve ser adicionado à zona hierarquicamente superior para assim validar as chaves utilizadas para assinar a zona. O keyset contém as chaves utilizadas pela zona assinada, quer seja a ZSK, KSK, ou ambas.

Por último torna-se necessário explicar o modo de criação de um SEP, na nossa rede de resolvers DNSSEC, para tal, é preciso introduzir no nosso ficheiro de zonas .pt, o registo DS, juntamente com a definição do name server do porto.pt. É também importante referir que muita das opcções aqui implementadas se baseiam na norma [14].

```
$ORIGIN porto.pt.

; definition of the name servers for the sub-domain
@       IN      NS      ns01.porto.pt.
@ IN DS 15852 5 1 AB0A781C20EAFA36D4453BEDF4012B1A80157FF1
```

Outro método alternativo para a inserção de um SEP é a introdução no ficheiro named.conf do campo trusted-keys, colocando lá a chave que assina a zona, neste caso a KSK:

```
trusted-keys {
porto.pt. 257 3 5 AwEAAdi9pVL/+MryygaaejPnmZ+qJekp/aCcg6RXXZbywe
BRhatdgKoaKG/8xZty/Y6e1NEvV2e059qxusB1urDB/Zy4sKvvfO9ngUr29aACV9Y5zPsw
aXpiRObTEGn2GzbM2duE02tISgyHwUSgnqSGnEJ+IJP8/HjWIDceaQRBj+5P; }
```

Desta forma, conseguiu-se montar um sistema de testes, nomeadamente de resolução de nomes que indica sempre uma flag de dados autenticados.

O conceito utilizado nesta implementação assenta na ideia de existir uma ilha de confiança com a base no domínio .pt e que garante 'segurança' a todos as zonas/domínios que desta dependem *(child)*.

Deve-se ainda notar que para uma gestão mais fácil das zonas de DNS foi necessário criar regras lógicas, assim, cada ficheiro de zona ficava dentro de uma pasta com o nome do domínio, dentro dessa pasta existem mais duas pastas, a pasta ZSK e a pasta KSK, destinadas a guardar as chaves ZSK e KSK da zona, respectivamente.

Dentro da pasta da zona poderia existir um ficheiro com os registos ScalSec (tipo H ou K) da zona com o nome hradiusfile.ds.

Foi ainda criado o ficheiro keys.conf na pasta principal do DNS *(/etc/bind)* o ficheiro keys.conf que contém as chaves todas que se deseja associar ao serviço de DNS, sejam elas para o uso do DDNS(nsupdate) ou para o uso do *Transaction SIGnature* (TSIG).

5.2.2 Novas Ferramentas

- **getCaKey**, este programa é utilizado para obter a chave pública do certificado do nó, verifica se o certificado passado no primeiro argumento é de facto uma CA, obtém o resumo *(hash)* da chave pública e escreve de seguida para o ficheiro especificado no segundo argumento do programa.

```
Usage:

getCAPKey <Certificate Name> <Pubkey file> <Digest Type>(optional)
Certificate Name - CA Certificate from wich pubkey will be extracted!
Pubkey file - File to write our CA public Key!
Digest Type - SHA1, SHA256, SHA512
```

- **getHrr**, este programa é utilizado para obter os registos H de um FQDN ou IP. Útil para a execução de debug e verificação dos registos H que se encontram no servidor de DNS.

```
Usage:

getHrr --help <H RR hostname> --verbose(optional)
--help - Print this help
H RR hostname - Host whose H RR we want to retrieve
resolver - Optional resolver path (IP, or hostname) [To Be Done]
--verbose - more verbose operations
```

5.2.3 Scripts de Gestão

Os scripts de gestão foram criados em linguagem BASH e utilizam utilitários da shell linux standard(coreutils) bem como as novas ferramentas criadas. Além disso, depende também do serviço crontab para uma execução periódica dos scripts. Os scripts fazem validações da agenda tendo verificações para os dias do mês a que são executadas as actualizações, bem como as horas a que é iniciado o processo. Além dessas verificações, também permite de forma automática utilizar o comando sudo na execução das tarefas, pois a maioria das accções necessita de permissões root. Estes scritps permitem ao utilizador um nível de abstracção maior não tendo necessidade de consultar manuais dos outros utilitários.

- **dnsupdate**, é o script principal de gestão que permite a instalação dos diversos scripts e permite de forma ordenada a excução correcta dos outros scripts. Depois de executado pela primeira vez, instala os restantes scripts de gestão e insere também, uma linha no crontab que torna todo o processo de actualização de registos automático. O directório onde são instalados os ficheiros de gestão é controlável editando váriavel CERTOVERLAY no topo do scripts.

```
Usage:

dnsupdate <dns zone>
dns zone - Specifies to wich dns zone are the maintenance tasks
It criates rules and tasks for a first time use
It also supports multi zone maintenance
Other variable options:
 $DEBUG - Enable Debug!
```

- **dnsupdate-daily**, script responsável por actualizar, bem como o reassinar da zona passada no primeiro parâmetro. O intervalo de tempo com que este script corre depende do crontab. Após a instalação do ficheiro de zona é inserida uma entrada no crontab que utiliza o intervalo de tempo definido no cabeçalho do ficheiro.

```
Usage:

dnsupdate-daily <dns zone>
<dns zone> Specifies to wich dns zone are the daily maintenance tasks
```

- **dnsupdate-monthly**, este script actualiza os diversos parâmetros da zona, como por exemplo, a chave do nó e a chave de raiz de CA, sendo executado mensalmente. Actualiza também o crontab para a execução das próximas fases de *rollover* dos registos, para mais informações consulte a secção 5.2.4 no qual é explicado melhor o processo de *rollover*.

```
Usage:

dnsupdate-monthly <dns zone> <task>
dns zone - Specifies to wich dns zone are the daily maintenance
task - It can be any of the 3 tasks (0,1,2,3)
```

- **gen_key**, script que permite de forma simples gerar chaves para as zonas especificadas, de elevada versatilidade permite gerar chaves KSK ou ZSK para a zona especificada no primeiro parâmetro.

```
Usage:

gen_key <zone name> <key type> -a <algorithm type> -d <depth bits>
key type - specifies if the key is of type: KSK | ZSK
algorithm type - can be of the following types
        RSA | RSAMD5 | DH | DSA | RSASHA1 | HMAC-MD5 | HMAC-SHA1 |
        HMAC-SHA224 | HMAC-SHA256 |  HMAC-SHA384 | HMAC-SHA512
depth bits - bits depth of signing key [512...4096]
Other variable options:
 $DEBUG - Enable Debug!
```

- **nsupdater**, script que permite diversos tipos de accções nas zonas de DNS específicadas. Permite o *rollover* das chaves ZSK ou KSK assim como a substituição das chaves H, actualizações de registos ou a remoção de registos obsoletos.

```
Usage:

nsupdater <zone> <key type> <status> <options>
Zone - Dns zone to be updated
Key type - type of key updated, it can be:
           KSK | ZSK | H | R
status - key status, old | new
options - options to be applied to that key:
          add | del | change
Other variable options:
 $DEBUG - Enable Debug!
```

- **sign_zone**, o script permite assinar a zona especificada no primeiro parâmetro, para tal, é também necessário passar as chaves de KSK e ZSK da zona, sendo o tempo de expiração opcional. Suporta os seguintes tempos de expiração, 12h, 24h e 36h.

```
Usage:

sign_zone <zone name> -k <ksk location> -z <zsk location> -e <expiration time>
k - KSK file location
z - ZSK file location
e - Signature expiration time
Other variable options:
 $DEBUG - Enable Debug!

Please note that you need to have a 'regular' directory map
```

- **binddsupdater**, script destinado a facilitar a troca de chaves, cria as condições necessárias para que o servidor RADIUS coloque os registos numa pasta pré-estabelecida. Este script corre no boot, como serviço e cria as pastas destinadas a receberem remotamente os ficheiros, a partir daí os registos serão tratados por outros scritps.

```
Usage:
binddnsupdater <options>
Update H rr set in pre determined zones
options -> create | delete | bind
Other variable options:
 $DEBUG - Enable Debug!

Please note that you need to have a 'regular' directory map
```

5.2.4 Períodos de Actualização e Revogação

Uma das fases deste trabalho que mereceu um pouco mais de reflexão foi este capítulo, uma vez que é necessário o correcto planeamento das mudanças (*rollovers*) das chaves e certificados para evitar possíveis problemas no DNS e na autenticação das estações. A seguir é descrito o processo utilizado para fazer o *rollover* dos certificados. São também apresentadas soluções alternativas e comparação das mesmas com a solução escolhida.

5.2.4.1 Processo de Actualização do RADIUS e do DNS

O processo de actualização dos certificados do servidor de autenticação está separado em três fases distintas. Na primeira é gerada o certificado do nó para o servidor de AAA, na segunda fase, é feita a actualização do certificado do nó e de seguida é feita a actualização do registo H a que o certificado estava associado, por último, a remoção dos registos e certificados obsoletos.

O modo de implementação do processo de actualização é planeado de forma a existir uma separação lógica e física dos serviço de AAA, do serviço de DNS. Assim, dois métodos surgem como alternativas válidas na actualização dos registo de DNS, por método remoto ou por método "local". Se se optar pelo método remoto é utilizada a funcionalidade de DDNS, se se utilizar

o método local, as zonas no servidor de DNS são actualizadas localmente e o servidor de DNS recarrega as configurações da zona sempre que necessário. Este último é de execução um pouco mais complexa pois a sincronização dos registos locais pode dar origem a outros problemas que o método remoto não sofre. Possui no entanto outras vantagens que tornam este método mais atractivo.

Optou-se por utilizar o método ”local“, que tem como maior vantagem no reínicio do serviço de DNS (quer seja um reinício administrativo ou outro) não perde o último registo válido, garantindo-se que os registos vão-se manter em sincronia com os certificados presentes no servidor de autenticação, excepto nos casos em que a comunicação directa entre os dois não for possível. Aí o método remoto é um bom complemento do método ”local“.

Uma vez que os servidores se encontram físicamente afastados é necessário definir um método de actualização dos registos do serviço de DNS com os novos registos H sempre no servidor de autenticação um novo certificado do nó é gerado. Para a passagem dos registos H entre o servidor de autenticação e o servidor de DNS é utilizando o ssh, existem outras opcções mas esta foi a que se decidiu adoptar, pois o ssh é um protocolo seguro, amplamente usado e que permite a execução das diversas operações de forma transparente para o administrador da rede.

Nas fases seguintes, após a recepção dos novos registos, é feita a sincronia de registos da zona alvo, actualizando-os e reassinando todo a zona, sendo por último forçado um reload no servidor de DNS para que este recarregue a zona de DNS.

A implementação do processo da rotação dos certificados do servidor de autenticação consiste na criação diária do certificado do servidor (validade de 24h) e na mudança mensal do certificado do nó bem como na consequente geração de um novo certificado do servidor. Apesar de parecer ser uma periocidade curta, este facto, tem como objectivo permitir que os certificados tenham um baixo número de bits, tentando manter a porpocionalidade entre validade vs profundidade de bits regra sempre presente aquando do uso de chaves digitais.

Como, após a mudança do certificado do nó os registos H do servidor de DNS necessitam de ser actualizados, é necessário associar a geração de novos certificados à actualização dos registos H ao longo das cadeias de topo de DNS. Para este processo é necessário salientar, que apesar da sincronia necessária tratam-se de sistemas independentes. Assim, pode-se separar o processo de actualização mensal em três fases distintas, englobando nelas as accções a serem tomadas tanto no servidor de DNS como no servidor de autenticação:

Novos certificados/registos – Estas acções deverão ocorrer 2 TTLs antes do fim do prazo dos registos só assim se consegue garantir um *rollover* eficiente obedecendo às recomendações do IETF.

- Geração de novos certificados do nó para serem utilizados pelo servidor de autenticação.
- Geração dos registos H baseados nos novos certificados.
- Os novos registos são colocados nas zonas do DNS a que servidor de autenticação está associado.

- É gerada uma nova chave ZSK e adicionada ao ficheiro da zona. Os períodos de actualização das chaves de ZSK coincidiram com a actualização dos certificados do servidor de autenticação.
- É adicionado o novo registo H à zona, contendo ela agora dois registos H sendo reassinada com a chave ZSK *antiga*, tendo a validade de dois TTLs, ou seja, é mantida a validade do registo com a chave ZSK *antiga*.
- É forçado o recarregar das zonas ao servidor de DNS e assim propaga a zona ao servidores slave no próximo pedido.

Mudança de certificados/Assinatura nova – Um TTL antes do fim do registo são efectuadas as seguintes acções:

- Nesta fase o certificado do nó é mudado no servidor de autenticação, bem como o certificado do nó. A substituição é feita pelos certificados gerados na fase precedente.
- A zona de DNS é reassinada usando a chave ZSK gerada na fase anterior, possuindo a validade de 2 TTLs

Remoção dos certificados/registos antigos – As acções são executadas *on time*, ou seja, precisamente quando passou 1 mês após o lançamento do todo este processo.

- Os certificados antigos são removidos.
- Os registos H caducados são removidos e é actualizado o ficheiro da zona.
- A chave ZSK antiga é removida com a consequente actualização do ficheiro de zona.

Na figura 5.2 é possível identificar o período de validade no espaço temporal de um certificado através das linhas a cheio, enquanto que as linhas a tracejado indicam que o certificado é criado mas ainda não é utilizado, por fim, é também possível identificar uma linha mais fina que indica que o certificado ainda é válido mas já se encontra em processo de substituição. Esta figura também pode ser transposta para o *rollover* dos registos de DNS, no qual, a linha a cheio indica a validade da assinatura de um registo, sendo possível identificar os períodos de *rollover* em que existem dois registos em simultâneo. Pretende-se tornar a ilustração o mais perceptível e possibilitar a correcta identificação das fases em que ocorrem em cada ciclo.

Assim e analisando a escala é possível ver que no inicio do processo o certificado possui uma validade de 24h a partir das quais é iniciado o processo de mudança de certificados de raiz sendo para isso necessário dois períodos de TTL extra. Desta forma, o período de validade de um certificado tem que ter sempre este facto em conta, no momento em que o certificado é gerado deve ser adicionada à data de validade dois TTLs extra necessários para que a troca de registos de DNS seja feita de forma transparente e obedecendo às melhores práticas.

Um resumo com mais detalhes acerca das características dos certificados pode ser consultado na tabela 5.1.

A definição do período temporal de actualização dos registos de DNS reveste-se de extrema importância, pois, se o tempo definido for muito alto implica a utilização de chaves maiores, bem

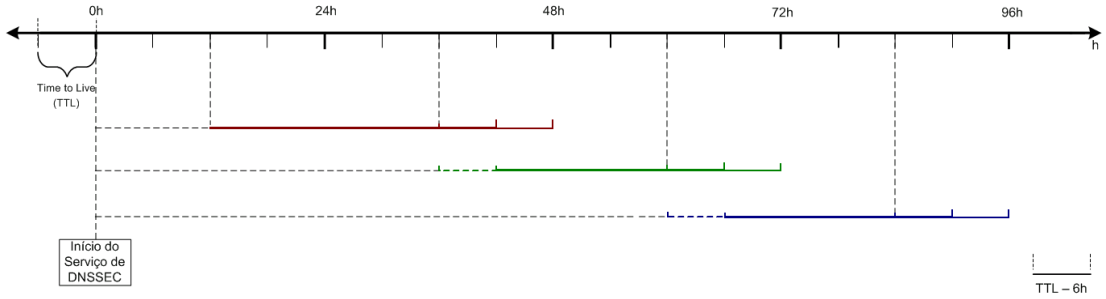


Figura 5.2: Escala cronológica de geração de certificados de raiz de CA (caso tenha actualização diária)

Tipo de Certificado	Profundidade	Duração
Certificado de raiz de CA	512 bits	1 mês
Certificado de nó	384 bits	1 dia

Tabela 5.1: Detalhes dos certificados utilizado pelo servidor de RADIUS

como, um aumento do tamanho das zonas assinadas, por outro lado se o TTL for muito baixo, gerar-se maior tráfego no servidor de DNS. Assim, decide-se definir o valor do TTL para 6h, este período de tempo é o que melhor se adapta à profundidade da chave (ZSK) vs entropia da chave (ZSK), cumprindo ainda todas as recomendações dadas no manual de boas políticas na aplicação do DNSSEC [14]. Sendo por isso necessário, que o TTL de um registo tenha uma fracção do tempo do registo, evitando assim, uma maior congestionamento de tráfego por parte dos servidores de DNS secundários ”slaves“. A validade dos registos de zona é de 24h ao fim das quais são reassinados, ao definir-se o TTL com 6h, consegue-se partir o dia em quatro TTLs o que facilita a gestão das zonas de DNS.

Os períodos de mudança da chave ZSK do DNS coincidem com os períodos de revogação dos certificados de raiz de CA do servidor de autenticação para facilitar e agilizar o processo de gestão permitindo desta forma que os actos são executados de uma vez só, sem desperdicio de recursos, doutra forma seriam feitas muitas mais assinaturas criando grande latência na correcta propagação dos registos.

5.2.4.2 Processo de RollOver usado no DNS

O processo de *rollover* da chave de ZSK implementado neste trabalho, segue os passos identificados na figura 5.3.

As acções na figura 5.3 foram extrapoladas do manual de boas políticas na aplicação do DNSSEC em [14], são explicitados um conjunto de regras a ter em conta quando se usa o DNSSEC. Algumas das regras lá existentes já foram abordadas no decorrer deste capítulo, no entanto existe outra que quero referenciar, prende-se com o tempo do TTL que deve ser suficientemente grande para garantir que todos os RRs podem ser validados ao longo de toda a cadeia, isso garante a sanidade dos registos de uma zona permitindo validar essa zona em tempo útil.

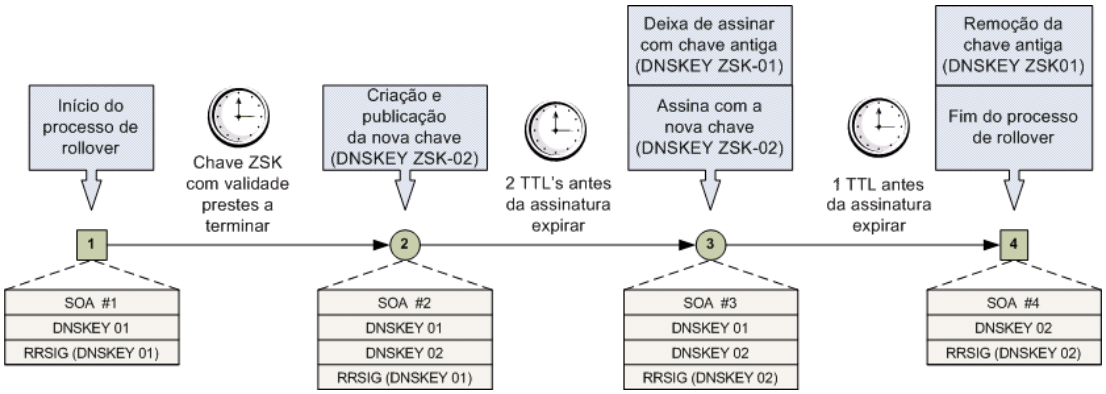


Figura 5.3: Accções executadas no rollover de uma chave ZSK

Este processo de *rollover* tem o nome de Pré Publicação da chave (*"Pre-Publish Key"*) e permite que a nova chave seja publicada mais cedo para os servidores de DNS secundários, necessitando por isso de quatro passos.

Na tabela 5.2 é possível identificar as chaves e a sua duração para o sistema DNS idealizado.

Domínio	Tipo de Chave	Profundidade	Duração
. (root)	KSK	4096 bits	1 ano
	ZSK	2048 bits	6 meses
.pt	KSK	1536 bits	1 ano
	ZSK	1024 bits	3 meses
porto.pt	KSK	1280 bits	1 ano
	ZSK	1024 bits	1 mês

Tabela 5.2: Rotação das chaves KSK e ZSK do sistema de DNS idealizado.

Um método alternativo de *rollover* que pode ser usado é o descrito na figura 5.4. A este tipo de *rollover* é referenciado no manual [14] como de dupla assinatura (*"Double signature"*).

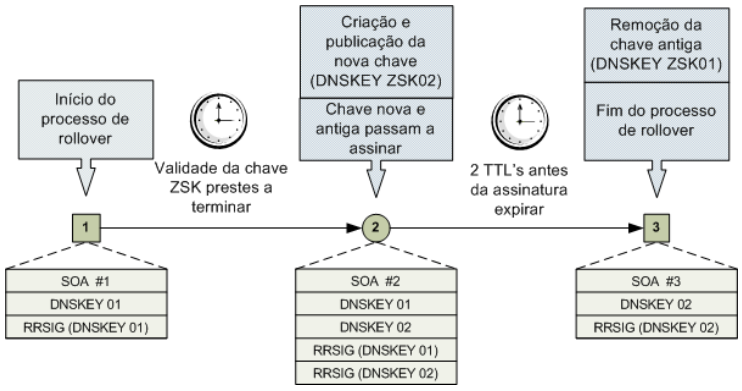


Figura 5.4: Accções executadas no rollover alternativo de uma chave ZSK

Para isso deve-se seguir os passos definidos na figura 5.4, como é possível verificar este tipo de *rollover* implica menos um passo que no outro *rollover*. A assinatura da zona é executada duas

vezes durante o *rollover*, necessitando por isso de apenas 3 passos. No entanto, este facto faz com o tamanho da zona aumente consideravelmente, tornando-se uma solução indesejada em zonas muito grandes devido às complicações que pode provocar durante essa fase.

Todos estes processos de *rollover* referencem-se às chaves ZSK, nas quais qualquer um dos processo de *rollover* pode ser usado sem que nenhum problema ocorra, no caso das chaves KSK é necessário ter em conta que ao ser feito o *rollover* a "zona hierarquicamente superior" necessita de ser actualizada (RR DS), bem como outros SEPs, isso demora algum a propagar-se ao longo dos servidores deve-se ter presente um factor de atraso.

Com o nosso servidor de autenticação e de DNS configurado, é agora necessário proceder-se às alterações no *supplicant* da estação.

5.3 Supplicant

O wpa_supplicant é um programa que permite a gestão e autenticação de estações que necessitem de um *supplicant* compatível com o 802.1X. Permite a negociação de chaves, com um servidor de autenticação, assim como, o controlo de roaming, autenticação e associação a APs.

O wpa_supplicant é constituído por diversas partes, o módulo da configuração, o daemon, a máquina de estados, os módulos de acesso ao interface e os módulos de autenticação, tendo ainda um GUI (configuração visual) e um cli (configuração na shell). Ao se iniciar o wpa_supplicant vários módulos são carregados. O módulo da configuração permite a validação do ficheiro de configuração e carrega a estrutura de dados necessários para o daemon funcionar; o daemon, corre em background e comunica com todos os módulos carregados; a máquina de estados é responsável por chamar os diversos métodos de autenticação, verificando o resultado dos mesmos; o módulo de acesso ao interface, é responsável pelo controlo dos interfaces, reconhecendo as características de cada driver e os módulos de autenticação que são carregados enquanto o programa decorre, permitem a execução do protocolo de autenticação carregado pela máquina de estados.

5.3.1 SSID Global

Após a abordagem teórica sobre a solução encontrada descreve-se o trabalho prático desenvolvido para suportar o *SSID global*.

Para tal configurou-se uma testbed, para suportar o nosso servidor RADIUS, atribui-se um IP estático ao nosso AP, ser possível acedê-lo com facilidade, configurando os restantes parâmetros via DHCP.

Arquitectura da rede montada, na testbed.

O material utilizado para esta testbed e em todo o processo de demonstração de resultados, é o seguinte:

- Estação portátil - Notebook Lenovo R61
  - Processador Core2 Duo CPU T7100

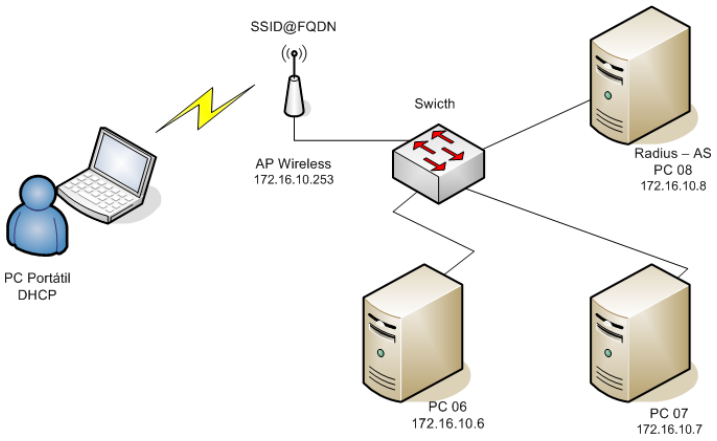


Figura 5.5: Esquema montado para a solução descrita na secção 4.1

  - 1GB de RAM DDR2
  - Placa de rede wireless intel 4965AGN
  - Linux 2.6.23 - Gentoo 64 bits

- AP - Cisco Aironet 1200 series
- Switch - Cisco Catalyst 2900 XL
- Servidores de DNS / RADIUS - Compaq Presario P4 / 1Gb de RAM (Debian 32 bits)

5.3.2 ScalSec

A segunda solução é a resposta a um problema complexo e de difícil resolução, é implementado um método de certificação que permite validar os certificados do nó com o ScalSec.

Na implementação da solução é feita uma separação lógica do código desenvolvido, através do uso da variável *CONFIG_SCALSEC* e de marcas de pré processamento, permitindo dessa forma habilitar ou desabilitar o suporte da solução desenvolvida, aquando da compilação do supplicant.

```
#ifdef CONFIG_SCALSEC
....
#endif /* CONFIG_SCALSEC */
```

Para suportar ambas as soluções, nomeadamente, a segunda solução é necessário adaptar o que se tinha executado na primeira solução. As diferenças são diminutas mas no entanto, notórias. Já não é obrigatório o uso de um certificado do nó para a verificação dos certificados.

Outra diferença é na comparação do *Alternative Subject Name*, uma vez que na primeira solução é o FQDN, na segunda solução é apenas a palavra <tipo>:radius.

Quando executado o *supplicant* através do uso libiptc comunica directamente com a firewall e bloqueia todo o tráfego que entra ou sai da estação, durante a autenticação EAP são verificados os certificados, sendo que não é feita a verificação do certificado do nó. Sendo apenas validado

se a cadeia de certificados é válida e se os certificados individualmente são ”válidos” (validade, extensões, formato). Sendo guardada toda a informação necessária para a validação do certificado do nó numa estrutura de dados do tipo ca_scalsec.

```
struct ca_scalsec {
        /* ca_pkey - ScalSec Public key from TLS CA wpa_supplicant receives
         * this key when validates the certificates exchanged in a TLS connection
         */
        const char *ca_pkey;
        /* ca_pkey_len - ScalSec Public key length
         */
        size_t ca_pkey_len;
};
```

O próximo passo, é feito através da execução de um pedido de DNSSEC do FQDN presente no ficheiro de dados ScalSec, ao resolver que deve responder com os registos H atribuído àquele FQDN. Se as chaves coincidirem com algum dos registos obtidos então a autenticação pelo sistema ScalSec é bem sucedida e são repostas as regras de firewall na estação, abrindo assim o tráfego à estação.

Diagrama de decisão da regras de verificação dos certificados, na Figura 5.6.

Assim, basta que se esteja a autenticar perante uma rede com *SSID global*, para se efectuar a verificação dos certificados.

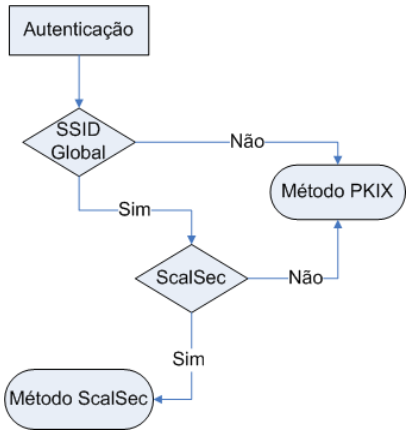


Figura 5.6: Lógica de decisão

```
} else {
    /* No ca_cert configured */
    if (conn->peap_sepchar) /* SSID' Separator */
        SSL_set_verify(conn->ssl, SSL_VERIFY_NONE, tls_verify_cb);
   else
        SSL_set_verify(conn->ssl, SSL_VERIFY_NONE, NULL);
}
```

É necessário também verificar se os certificados verificam as regras ScalSec, se tal acontecer, passar relativa aos certificados para a estrutura de dados *ca_scalsec*.

```
if (os_strstr(match,"CN=ScalSec")) { /*Verificar se é a arquitectura ScalSec */
     wpa_printf(MSG_DEBUG,"CB: ScalSec match specified");
.....
     tls_match_Scalsec(err_cert); /*only CA*/
```

Caso ocorra algum erro durante a autenticação ScalSec é dispoteletado um aviso e é apagada toda a informação da autenticação que está retida na estrutura *ca_scalsec*.

```
if (depth == 0 && !tls_match_altsubject(err_cert, MYALTSUJECT)) {
     wpa_printf(MSG_WARNING, "TLS: altSubjectName match"
         "'\%s' not found", MYALTSUJECT);
         preverify_ok = 0;
         set_error();
         temp=NULL;
```

A estrutura de dados que é passada à thread é constituída pelo identificador da thread, pelas estruturas que contém as informações dos certificados e registos de DNS obtidos, bem como a hash do certificado do nó.

```
struct scalsec_ops {
       uint tscalsec_res; /* Resultado da thread */
       pthread_t Scalsec_thread; /* Identificador da thread */
       struct ca_scalsec *pscalsec; /* Informação da CA */
       struct dsrr_type *hresult; /* Registos RR's do DNSSEC */
       u_char sha1sum[HASHSIZE]; /* Digest do certificado */
};
```

No fim da negociação de EAP é lançada uma thread identificada pela variável *Scalsec_thread*, através da qual iría-se verificar se as credenciais do NAS são válidas. A primeira passo a tomar é garantir que mais nenhum tráfego saía da estação a não ser o tráfego de DNS para a obtenção dos registos H. Para tal, obtia-se um backup das regras que estavam presentes no iptables e limpavam-se as regras existentes. De seguida, alterava-se a política das *chains* para fazer DROP de tudo e por fim apenas se habilitava a passagem do tráfego de DNS.

```
       chaines=itptables_backup(&h);
.....
       wpa_printf(MSG_DEBUG, "Flushing entries");

       flush_iptables(&h);
.....
       if ( ! allowdns(fwchain) )
```

Depois da firewall ter sido configurada com sucesso obtia-se os registos H da localização especificada no *SSID global*, para a validação dos registos é suportado um FQDN ou IP.

```
if (type[0] == 1)
        scalsec_ops.hresult=get_ds_by_ip(hostname);
else if (type[1] == 1)
        scalsec_ops.hresult=get_ds_by_fqdn(hostname);
```

Se a comparação for bem sucedida então a thread termina, continuando a ligação activa, se os certificados não forem válidos, o *supplicant* desassocia-se do AP e tenta encontrar outro válido.

```
if ( scalsec_ops.tscalsec_res == 1 ) {
        wpa_printf(MSG_DEBUG, "ScalSec successful");
}
else {
        wpa_printf(MSG_DEBUG, "ScalSec failed, no match");
        wpa_supplicant_disassociate(wpa_s, WLAN_REASON_UNSPECIFIED);
}
```

Foram ainda desenvolvidas diversas funções com o objectivo de verificar se os certificados seguiam as regras ScalSec, como por exemplo, a função tls_match_ScalSec(), que tem também por fim extrair o valor da chave pública do certificado do nó e para tal, executa a função tls_get_pubkey().

Procedeu-se ainda a uma alteração no modo de validação da redes Wi-Fi que não permite o uso de protocolos de autenticação sem cifra, por exemplo PAP, aquando do uso da flag peap_sepchar. Seguindo-se algumas das recomendações do IETF em [24].

```
if (ssid->peap_sepchar &&
        (ssid->phase1 ? os_strstr(ssid->phase1,"auth=PAP") : 0 ||
        (ssid->phase2 ? os_strstr(ssid->phase2,"auth=PAP") : 0 ))
    wpa_printf(MSG\_ERROR, "It's not possible to set PEAP method and set"
        " authorization to PAP");
    errors++;
```

Para a verificação desta solução montou-se na testbed a seguinte arquitectura da rede.

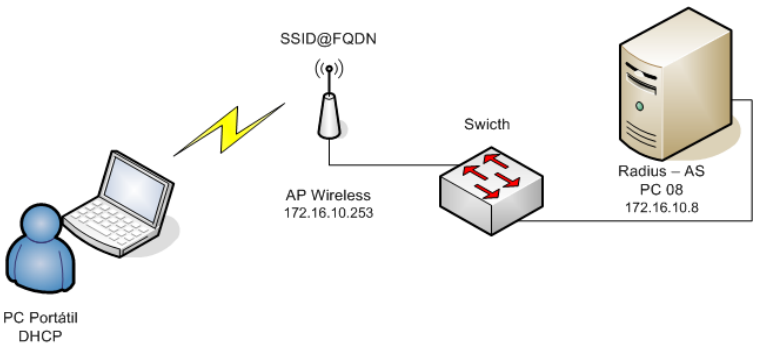


Figura 5.7: Esquema montado para a solução descrita na secção 4.2

5.3.3 LWRESD

Outro dos pontos a ser tratado era a obtenção dos registos do DNSSEC, entre as muitas bibliotecas/programas disponíveis decidiu-se optar pelo *Lightweight Resolver Daemon* (LWRESD). O LWRESD é um daemon que é distríbuido juntamente com o BIND do *Internet Systems Consortium* (ISC), tem diversas vantagens quando comparado com o BIND e é a melhor solução se se deseja criar um servidor de DNS com poucas zonas e sem configurações complexas. No entanto, também pode funcionar como um resolver, possui um ficheiro de configuração que permite configurar as opcções avançadas dando assim uma flexibilidade difícil de alcançar com qualquer outra biblioteca dinâmica de DNS. Assim um dos pontos fortes do LWRESD é a sua versatilidade e a facilidade com que se consegue configurar o resolver a utilizar opções avançadas como o DNSSEC ou o TSIG. Outra vantagem advém do facto de utilizar parte das bibliotecas partilhadas do BIND permitindo dessa forma ter funcionalidades avançadas que doutra forma seriam difíceis de existirem noutros resolvers. O uso de um resolver que é um daemon, estando por isso sempre a correr em background, possibilita outra vantagem que é o facto de passar a existir cache. Desta forma os pedidos de DNS serão resolvidos mais depressa pois em alguns casos não existe a necessidade de pedir a informação a resolvers externos pois a informação consta na cache interna do resolver. Por último e o que se considerou como uma das vantagens mais decisivas é o facto de LWRESD ter uma API simples e o esforço necessário para implementar esta solução é bastante reduzido quando comparado com as outras opcções. Por exemplo, o LWRESD permite de uma forma simples substituir a API de referência do DNS que utilize o BIND pois é uma API compatível. Ganhando dessa forma ainda uma outra vantagem que é a existência de inúmeros exemplos para o resolver do BIND que podem ser aplicados quase directamente ao LWRESD. Os resultados que a API retorna dependem da forma como o daemon está configurado, assim se o daemon estiver configurado para suportar DNSSEC os resultados virão de um determinado modo, mas se o DNSSEC estiver desabilitado os resultados returnados já serão diferentes. É também importante referir que é possível configurar o comportamento do LWRESD através da API sem se necessitar editar manualmente o ficheiro de configuração. O uso desta API nesta implementação baseou-se no uso da função getrrsetbyname, que permite obter um conjunto de registos (RR) de determinado tipo e classe para o FQDN especificado.

5.3.3.1 Configuração

O LWRESD instalado na estação necessita de ser configurado, os comandos são muito parecidos com os comandos usados no servidor de nomes BIND. Neste primeiro bloco de configurações é demonstrado como configurar o LWRESD para fazer as necessárias verificações de DNSSEC e efecutar a cache dos registos. É adicionada a opcção de forwarders usando o servidor de DNS *(.pt)* como o servidor de nomes recursivo, assim se a estação não possuir um dado registo na cache local consulta o servidor de nomes recursivo configurado.

```
options{
        directory "/var/cache/bind";
```

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

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

É necessário introduzir o SEP no LWRESD indicado qual a chave em que se pode confiar. É possível verificar que os registos vindos do domínio *.pt* e que venham assinados com aquela chave serão considerados válidos e fidedignos.

```
trusted-keys {
"pt." 257 3 5 "AQOv3wVb2k6LsSz/56jL60kCQ2eke6YcFksI7cagUiBA/AwuJyb8X0Ii
              g09EcZ1SSAulNB6oQLNIdZviX5+lD8G4LWkLH+bwYrhm\dfrac{}{}KILn5qc6WuTA
              tIKHwgtAH+x3e+Tt8ps=";
};
```

Por fim, este passo é opcional e permite que o tráfego entre o LWRESD e o servidor de DNS vá cifrado usando o TSIG. O TSIG é baseado em Message Authentication Codes(MAC) e permite autenticar os pedidos e respectivas respostas com uma validade temporal relativa. A transacção segura dos registos é necesária pois permite adicionar outra camada de segurança ao transporte dos registo tornando qualquer hipótese de falsificação muito mais díficil. Para isso é necessário definir uma chave simétrica comum no servidor de DNS e no resolver, sendo também necessário adicionar a directiva *keys* no resolver identificando o servidor de DNS com o qual o tráfego é cifrado.

```
key transport.porto.pt.{
        algorithm hmac-md5;
        secret "3e9ekyhfgF0alDVi36EfdfNA==";
};

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

Assim todo o tráfego que o resolver enviar assinado com esta chave e tenham como destino o servidor de nomes *172.16.10.7* virá assinado com a mesma.

5.3.3.2 Funções Desenvolvidas

Foram desenvolvidas um conjunto de funções que permitem cumprir a obtenção dos diversos registos H do DNS. De seguida, é feita uma descrição sumária dessas funções assim como

dos valores que elas retornam tornando possível assim documentar e analisar melhor o código desenvolvido para este módulo.

uint is_ip(const char *hostname) – Esta função é uma função "helper"(suplementar) que tem como objectivo verificar se o array de chars passado à função é ou não um IP. Retorna o valor de 1(TRUE) se for um IP ou 0 se não for.

uint decoderr(struct rdatainfo *chunk, size_t bytelen, struct dsrr_type *dsregister) – Extrai os registos das pesquisas de DNS que estão de um modo raw (informação toda junta) e coloca-á numa estrutura de registos H.

uint free_hrr(struct dsrr_type *dsrr) – Função utilizada para libertar da memória a estrutura de registos H passada à função pelo apontador dsrr.

int get_ds_by_fqdn(const char *hostname) – Função que se destina a obter um conjunto de registos H para um determinado FQDN, verifica também se estes são válidos e retorna 1 (TRUE) se os registos existirem e se forem válidos, 0 para qualquer outro dos resultados possíveis.

uint get_ds_by_ip(const char *ip) – Esta função tem os mesmos objectivos que a função anterior só que utiliza um IP e não um FQDN. Utiliza para parte do seu processamento a função anterior (get_ds_by_fqdn) e retorna 1 caso obtenha algum registo e se estes forem válidos ou zero para os restantes resultados possíveis.

int strchrcount(const char *str , char cmp) – Função de uso geral desenvolvida para complementar as funções de strings disponíveis na GLibc, utilizada para contar o número de vezes que um determinado caracter existe num array de chars.

uint check_if_ip(const char * buffer) – Esta função verifica se o array de caracteres é ou não um IP e utiliza a função helper 'is_ip' para verificar o array. Retorna 1(TRUE) em caso de uma verificação bem sucedida e zero não sendo um IP.

int valid_hostname(const char *name) – Esta é uma função "helper" que verifica se o array de caracteres passados para a função é um hostname com uma construção válida, indica que o array é válido retornando 1 ou em caso de erro retorna 0.

uint check_if_fqdn(const char * buffer) – Verifica se o array de caracteres passado à função é ou não um FQDN e se é um FQDN com uma construção válida, retornan do 1 (TRUE) em caso de sucesso.

5.3.4 Iptables

O controlo do iptables é feito recorrendo a uma biblioteca interna do IPtables, a biblioteca libiptc. Com a inclusão desta biblioteca temos uma API estável de acesso directo aos módulos dos kernel, não havendo encapsulamento dos pedidos. Desta forma, o controlo sobre a firewall é mais

rápido e com menos *overhead*. A integração desta biblioteca nos trabalhos para a dissertação, teve os mesmos passos que o módulo do LWRESD, ou seja, primeiro é feita uma implementação standalone dando origem ao programa fw-editor, de seguida separou-se o código de forma modular de modo a poder-se integrar no supplicant. O uso da API e das funcionalidades da biblioteca libiptc é quase completa, pois o trabalho necessário aplicar ao nível de firewall é bastante complexo. As tarefas necessárias exigiam por esta ordem, o backup das regras existentes, a limpeza das cadeias, a aplicação de regras predifinidas, aplicar novamente a limpeza de regras nas cadeias e por último aplicar as regras previamente guardadas. Para a execução destas tarefas foram desenvolvidas um conjunto de funções que serão descritas mais detalhadamente no próximo capítulo.

5.3.4.1 Funções Desenvolvidas

O conjunto de funções que permite controlar de forma eficaz a firewall (iptables), sendo de seguida feita uma descrição sumária dessas funções assim como dos valores que elas retornam, permitindo que como já foi escrito no LWRESD uma melhor análise do código desenvolvido.

int allowdns(struct chain_rules *fwrules) – A função tem como objectivo aplicar as regras necessárias ao iptables de forma a que apenas seja possível fazer a negociação por DHCP e fazer pedidos de DNS. Retorna zero caso não consiga aplicar as regras com sucesso. Neste momento as regras são hardcoded apenas permitem tráfego DNS.

int iptables_apply_chaine(struct chain_rules *chaines, iptc_handle_t *h) – Esta função destina-se a aplicar a cadeia de regras passadas à função na firewall da estação.

int flush_iptables(iptc_handle_t *h) – Esta função faz um 'reset' a todas as cadeias, removendo todas as regras (entries) nelas existentes. Retorna 1 em caso de sucesso e 0 em caso de erro.

struct ipt_entry * target_add(struct ipt_entry *entry, const char *target_name) – Esta função permite adicionar targets às regras(entries) passadas à função. Retorna uma nova regra com a indexação do novo target em caso de sucesso ou um apontador NULL em caso de falha.

struct ipt_entry * udp_match_add(struct ipt_entry *entry, struct ipt_udp *rule)
tcp_match_add(struct ipt_entry *entry, struct ipt_tcp *rule) – Permite adicionar às regras correspondências aos protocolos de TCP e UDP, bem como explicitar as portas de origem e destino dessa correspondência. Em caso de sucesso, uma nova regra (entry) é retornada com a indexação da nova correspondência ou caso falhe é retornado um apontador NULL.

uint free_chain_rules(struct chain_rules *chains) – Esta função destina-se a apagar da memória a cadeia de regras passadas à função pelo apontador chains.

5.4 Conclusões

Ao longo deste capítulo foi explanado o trabalho implementado. Foi realizada a configuração do servidor de autenticação, usando o software freeRADIUS. Para a configuração do módulo EAP do freeRADIUS foram utilizados certificados X.509, tendo-se procedido à criação dos mesmos. No sentido de optimizar o processo de criação dos certificados foi desenvolvido um conjunto de scripts que permite a criação dos certificados, seguindo o modelo de certificação ScalSec, de forma rápida e automática.

Configurou-se o servidor de DNS com suporte para o DNSSEC e para aceitar chaves TSIG as quais foram utilizadas para a validação, comparação de resultados e teste do DDNS. Na implementação utilizaram-se boas práticas de gestão do DNS explicando-se pormenorizadamente a organização da sua hierarquia e ficheiros pertinentes, optimizando-se assim todo o processo de gestão.

Foram ainda demonstrados os procedimentos de *rollover* dos certificados e registos, quando utilizado o ScalSec, procedimentos estes, aplicados recorrendo a scripts que facilitam e automatizam a gestão de todo o processo de *rollover* e chaves.

Na implementação dos serviços não foi necessária qualquer alteração do código do servidor de autenticação. O *supplicant*, foi alterado por forma a que este suportasse o modelo PKIX e o modelo ScalSec, com ou sem o uso do *SSID global*, sendo que para este último foi necessário configurar o LWRESD e definir a API desenvolvida.

Por último, para o controlo da firewall utilizou-se a libiptc explorando a API criada.

Capítulo 6

Testes e Validações

Com vista a criar uma solução de segurança que permita às estações 802.11 identificar correctamente redes Wi-Fi, de forma a evitar redes forjadas, dividiu-se a solução proposta em duas complementares, nomeadamente o *SSID global* e a arquitectura ScalSec, uma vez que analisadas em separado permitem obter um maior detalhe sobre as vantagens e desvantagens de cada uma, comparativamente às soluções já existentes.

Torna-se assim pertinente proceder à análise crítica de resultados, por forma a avaliar de forma objectiva a qualidade das soluções desenvolvidas. A maior parte deste trabalho incidiu sobre a autenticação. Neste sentido, serão expostos neste capítulo os cenários nos quais as soluções são utilizadas, definindo ainda os parâmetros que se revestem de maior importância analisar. No que se refere aos requisitos necessários para a autenticação de uma estação identificam-se os seguintes: segurança, tempo de associação e tráfego requerido.

Identificou-se que a primeira solução, o *SSID global*, não envolvia mudanças significativas no protocolo de autenticação, sendo que as mudanças incidiram mais no *supplicant*. Por esse motivo apenas será analisado o tempo de autenticação, pois as alterações incidiram na inclusão de regras para o processamento dos certificados do servidor de autenticação. Por outro lado, como o tamanho dos certificados sofreu alterações insignificantes não se considera pertinente avaliar o tráfego gerado por essa solução. Por sua vez, na segunda solução, a arquitectura ScalSec, qualquer um dos requisitos identificados deve ser alvo de avaliação, pois o método de autenticação foi alterado, sendo por isso importante analisar se as modificações foram benéficas e se a solução é uma solução válida.

6.1 Cenários de Utilização

Os cenários de utilização aqui descritos são os cenários que uma estação móvel 802.11 encontrará aquando da sua conectividade wireless. Estes, foram separados com o propósito de ilustrar os diferentes casos de uso, sendo realçados os pormenores mais relevantes.

6.1.1 Primeira Autenticação

Neste cenário, a estação irá autenticar-se perante a rede Wi-Fi pela primeira vez recorrendo a uma autenticação 802.1X. Para tal, como já foi explanado anteriormente, recorre a um *supplicant* que irá proceder à negociação TLS. O *supplicant* cria uma nova sessão TLS para a troca de credenciais com o servidor de autenticação, a qual tem uma validade temporal e utiliza certificados de chave pública.

6.1.2 Re-Autenticação na mesma ESS

Após algum tempo activo na rede Wi-Fi a estação desconecta-se e passado pouco tempo deseja reconectar-se novamente. Desta forma, irá re-autenticar-se perante a mesma rede Wi-Fi. A abordagem a este cenário não é linear, mas as duas hipóteses mais comuns são as seguintes:

- O *supplicant* suporta a funcionalidade de re-autenticação TLS, ou seja, guarda a sessão TLS e resume-a para a troca de credenciais, sem que esta necessite de um novo handshake e consequente troca de certificados de chave pública. Esta funcionalidade é suportada pelo Wpa_supplicant e é conhecida como fast-reauth, no entanto, nem todos os protocolos EAP suportam esta funcionalidade.

- O *supplicant* não suporta, ou não está activa a funcionalidade de re-autenticação TLS, neste caso é necessário criar uma nova sessão TLS, assim como nova troca de credenciais.

6.2 Análise do Tráfego Gerado

Esta é uma das análises que se pode considerar mais importantes, pois com ela consegue-se verificar quanto *overhead* é que a solução desenvolvida introduz no processo de autenticação. Estão presentes no gráfico, três possíveis autenticações usando o ScalSec, a primeira é usando o modelo ScalSec "normal", na segunda é utilizado o modelo ScalSec mas o *resolver* DNS tem o suporte TSIG activado e por último o modelo ScalSec com o recurso à cache do *resolver* de DNS. No caso do PKIX são abordadas duas análises, uma utilizando o modelo PKIX normal, na qual o *supplicant* contém o certificado do nó localmente instalado e na segunda análise, a qual além da autenticação PKIX normal possui uma validação do estado do certificado do nó. Para a validação dos certificados a hipótese do uso de uma CRL foi descartada, pois normalmente tem um tamanho superior mas uma validade inferior ao do OCSP, como tal, a comparação foi executada recorrendo ao uso do OCSP. Infelizmente, a versão do *supplicant* tal como a versão do servidor de autenticação (RADIUS) não suportam a verificação OCSP durante o TLS, foi assim necessário estimar um valor que indicasse qual o impacto provocado no tráfego médio PKIX. Recorreu-se a diversos certificados teste e foram executadas várias experiências para que esse valor fosse o mais próximo possível do valor real.

OCSP	Número de amostras	Tamanho médio	Intervalo de confiança (95%)
Respostas	15	704,5 bytes	[671,5 ; 737,4]

Tabela 6.1: Tamanho médio de uma resposta OCSP

Tendo para isso sido consultada a norma do TLS em [7], na qual se constata que o valor passado na resposta TLS era equivalente a uma resposta OCSP. Assim, nos diversos testes conduzidos é apenas tomado em conta o tamanho do pacote das respostas OCSP e descartado tudo o resto. Pode verificar-se que o valor médio obtido para uma resposta OCSP é de 704,5 bytes, registando-se um aumento do tráfego médio no modelo PKIX para 5399,5 bytes, denominando-se esta amostra de PKIX+OCSP.

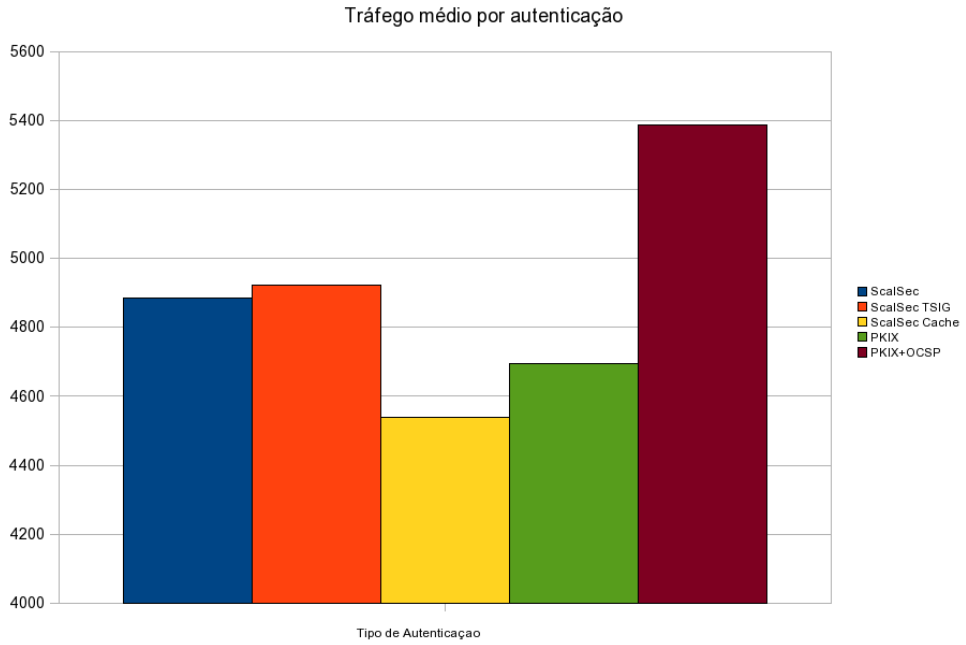


Figura 6.1: Tráfego médio por autenticação

Para uma melhor compreensão dos dados da figura 6.1, na tabela 6.2 é feita uma análise estatística desses mesmos dados, a qual, bem como todas as outras que neste capítulo serão apresentadas, tem como objectivo estimar valores que permitam oferir, com certeza, que os resultados obtidos nos cenários de uso acima descritos são válidos, tornando-a num valioso mecanismo de decisão.

Para a análise do gráfico da figura 6.1 ter-se-á os resultados do modelo ScalSec como valor base, sendo as comparações feitas a partir desse valor. Assim, na análise do modelo ScalSec com TSIG activo no resolver DNS é possível perceber que é uma das piores soluções em termos de tráfego médio gerado, sendo só ultrupassada pela solução PKIX+OCSP. O tráfego médio superior desta solução, provém do tamanho extra adicionado pela chave TSIG nas resoluções de nomes, uma vez que os certificados usados são iguais a todas as outras soluções ScalSec. No entanto, esta

	Número de amostras	Tamanho médio	Intervalo de confiança (95%)
ScalSec	4	4886 bytes	[4882 ; 4888]
ScalSec with TSIG	4	4924 bytes	[4921,4 ; 4926,6]
ScalSec with cache	4	4538 bytes	[4535,1 ; 4540,9]
PKIX	4	4695 bytes	[4690,4 ; 4699,6]
PKIX+OCSP	4	5399,5 bytes	[5386,9 ; 5411,6]

Tabela 6.2: Análise estatística do tráfego médio por autenticação

solução tem como vantagem uma maior segurança no transporte dos registo de DNS.
No caso do modelo ScalSec com cache identifica-se a mesma como sendo a melhor solução em termos de tráfego médio gerado, sendo que esta requer menos 348 bytes que a modelo ScalSec e 157 bytes que o modelo PKIX. Facto originado pelo tamanho dos certificados, o qual é inferior ao do modelo PKIX, bem como pelo facto de o resolver não necessitar fazer nenhum pedido ao servidor de DNS uma vez que já possui a resposta em cache. Após a análise deste gráfico pode concluir-se erradamente que a solução ScalSec gera mais tráfego por cada autenticação que o modelo PKIX. A verdade, é que para se poder comparar as duas soluções de forma correcta, na solução PKIX teria de se recorrer ao OCSP. Só desta forma se conseguia ter o mesmo nível de segurança apresentado pela solução ScalSec, assim e recorrendo à solução PKIX+OCSP é possível constatar que esta é bastante pior que qualquer solução ScalSec.

6.3 Análise dos tempos de Autenticação

Como foi referido no inicio do capítulo esta análise é comum às duas soluções, no entanto, para uma melhor compreensão opta-se por realizar a análise em separado.

6.3.1 SSID Global

Existem diversos factores que interferem numa correcta análise da autenticação, podendo eles serem externos ou internos. Por haver alguma aleatoriedade, no tempo que demora a placa wireless a dar início aos pedidos de autenticação, assim como, no tempo de processamento do pedido EAP por parte do AP, definiu-se que a forma mais correcta para evitar este tipo de problemas na análise, seria retirando os tempos quando já haviam sido trocadas algumas mensagens de autenticação, mais propriamente no início do pedido EAP-PEAP. Ou seja, a análise incidirá no tempo que o *supplicant* demora a autenticar-se, descartando-se assim todos os outros factores aleatórios que poderiam ter impacto na nossa análise. O que se consegue observar através da tabela 6.3 é que a introdução das regras de processamento dos certificados não contribuiram para um aumento significativo do tempo de autenticação, registrando-se alterações na ordem das milésimas de segundos. Estes valores vêm confirmar que esta solução além de melhorar a segurança da estação e permitir uma correcta selecção da ESS pretendida, não afecta o tempo de autenticação de um modo significativo, sendo vantajosa a adopção de uma estrutura lógica do SSID para uma rede

Wi-Fi.

	SSID	SSID Global
	48 ms	54,3 ms
	52,1 ms	45,1 ms
	42,7 ms	48,4 ms
	44,3 ms	41,3 ms
	40,9 ms	48,3 ms
Média	45,6 ms	47,5 ms
Intervalo de confiança (95%)	[44,3 ; 46,9]	[46,1 ; 48,9]

Tabela 6.3: Comparação dos tempos de autenticação

6.3.2 Solução ScalSec

A figura 6.2 representa os tempos que o *supplicant* demorou a processar toda a operação de autenticação. No caso do ScalSec está a ser descartada a segunda fase de autenticação, na qual é feito o pedido de DNS, estando apenas a ser avaliado o tempo de processamento dos certificados das diferentes autenticações.

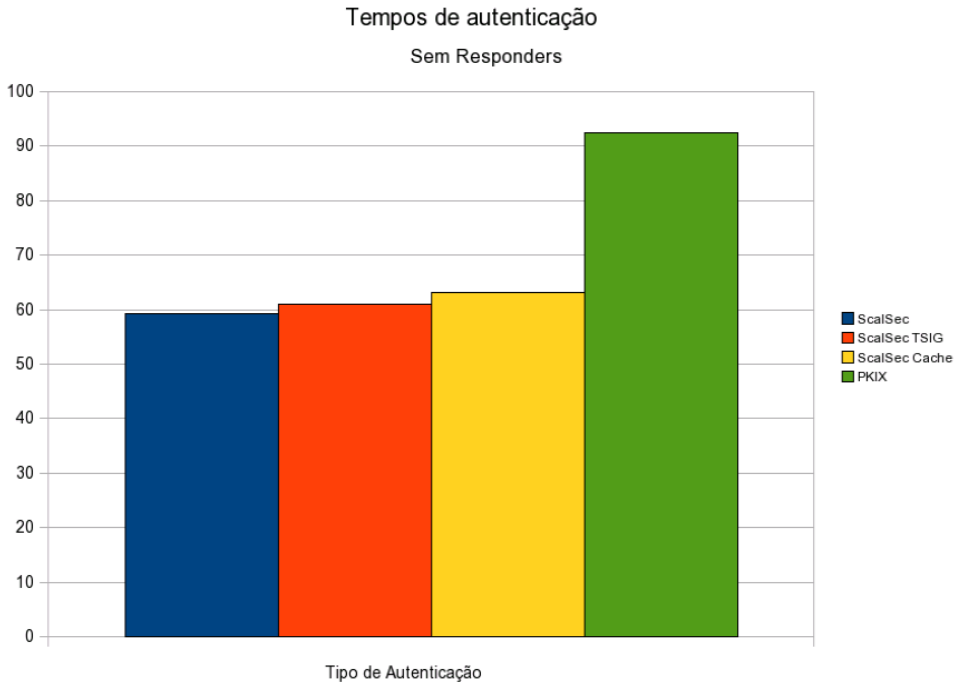


Figura 6.2: Tempos de autenticação em ms

Análisando a tabela 6.4 e tendo novamente os resultados ScalSec como base, verifica-se que a solução ScalSec possui valor mais baixo quando comparada com a PKIX, tendo a solução ScalSec TSIG um valor intermédio, correspondendo o valor mais alto à solução ScalSec com cache DNS.

	Número de amostras	**Tempo médio**	**Intervalo de Confiança (95%)**
ScalSec	4	59,3 ms	[57,3 ; 61,3]
ScalSec with TSIG	4	61,1 ms	[58,6 ; 63,6]
ScalSec with cache	4	63,2 ms	[60,4 ; 66,0]
PKIX	4	92,5 ms	[83,8 ; 101,2]

Tabela 6.4: Análise estatística do tempo médio por autenticação

Este valor não corresponde ao esperado, tendo em conta que se esperava que os tempos desta solução fossem iguais ou parecidos com a solução ScalSec. Por último é possível verificar que a solução PKIX possui o valor mais alto desta análise.

A verdade é que há uma diferença de tempos significativa que permite chegar a uma conclusão, a utilização da autenticação ScalSec é mais rápida do que a autenticação PKIX. Este facto fica a dever-se aos certificados ScalSec, que possuen chaves menores, tornando-se assim mais rápido o seu processamento, bem como ao facto de não se processar o certificado do nó, pois apenas numa fase posterior é feita a sua validação.

Com o objectivo de aproximar os resultados ao mundo real, deve-se ter em conta todos os outros procedimentos protocolares que as diferentes autenticações possuem. Assim no caso do PKIX, será tida em conta a análise de resultados usando o OCSP com e sem cache, através do valor médio obtido experimentalmente. No caso do ScalSec será usado um valor estimado por excesso.

Na tabela 6.5 é possível constatar o valor médio obtido dos tempos das respostas OCSP, sendo contabilizado o tempo desde que o pedido foi feito até ao momento em que a resposta chega à estação.

OCSP	**Número de amostras**	**Tempo médio**	**Intervalo de Confiança (95%)**
Respostas	15	116,6 ms	[106 ; 127,3]

Tabela 6.5: Tempo médio de uma resposta OCSP

Assim é possível verificar que o valor médio de uma resposta OCSP obtido é de 116ms, valor que o servidor de TLS demorará a enviar os dados à estação. O tempo médio dado para uma resposta DNS foi de 100ms, um valor estimado que se julga ser um valor médio próximo dos reais.

No caso do uso da cache em vez de se optar por uma solução ideal em que não haveria qualquer tipo de atraso, decidiu-se optar por reduzir os tempos dos *responders* para um terço do valor obtido/dimensionado, embora sejam esperados valores ainda mais baixos dos que foram aqui introduzidos.

A figura 6.3 é um gráfico que associa a figura 6.2 aos tempos médios das respostas dos diferentes *responders*.

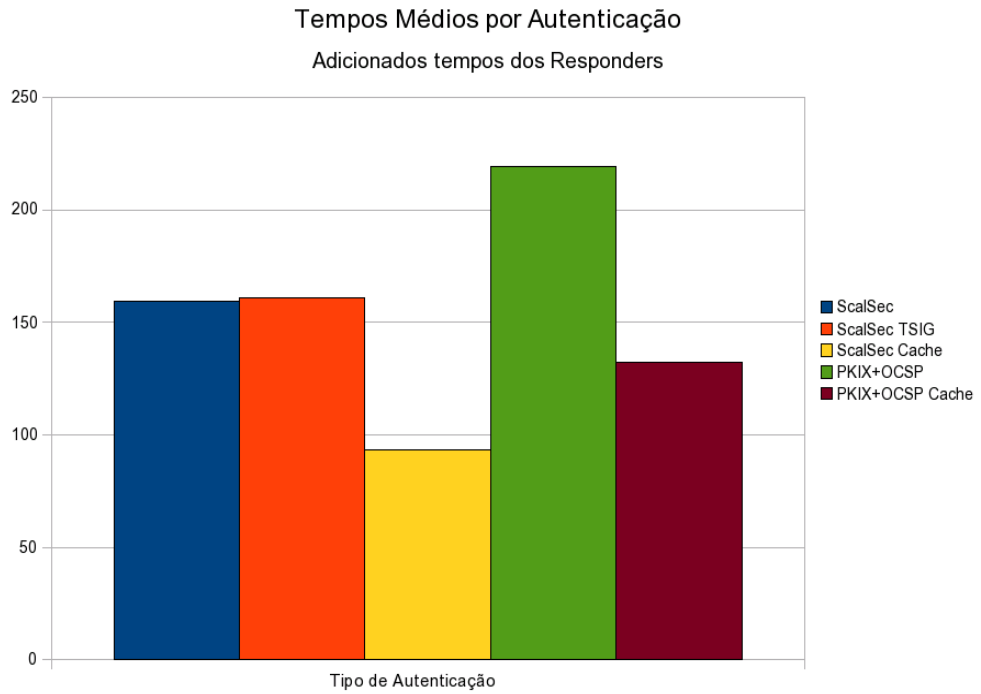


Figura 6.3: Tempos de autenticação somando o valor médio dos responders

O que é possível verificar na figura 6.3 é que os resultados não se alteraram substancialmente em relação aos da figura 6.2. Na figura 6.3 e mantendo o método de análise seguido no gráfico anterior é possível verificar que a solução ScalSec, bem como a solução ScalSec com TSIG possuem um valor intermédio, sendo estes mais baixos que a solução PKIX+OCSP. A solução ScalSec com o uso da cache de DNS volta a ter o valor mais baixo desta análise, sendo a solução PKIX+OCSP a pior solução. Com a introdução da cache no OCSP deverão ser esperadas melhorias nos tempos de autenticação, tendo esta solução registado o segundo melhor resultado.

Como se pode constatar no gráfico da figura 6.3 a diferença dos tempos diminui ligeiramente com a adicção dos tempos médios, mas mantém-se a vantagem para a arquitectura ScalSec. Apesar dos tempos médios dos resultados com cache estarem estimados, a solução ScalSec apresentará sempre um tempo médio de resolução de nomes inferior ao OCSP, pois embora o *responder* OCSP possa correr sob diferentes protocolos, como é o caso do HTTP ou FTP, utiliza sempre ligações TCP o que normalmente demora mais tempo do que o envio de pacotes UDP, como acontece no caso do DNS.

6.4 Conclusões

Neste capítulo foram identificados os diferentes cenários necessários para a validação da solução, bem como a análise ao tráfego médio gerado e tempos de autenticação.

Os resultados obtidos permitem validar as soluções apresentadas, no que se refere ao tráfego médio gerado por solução obteve-se resultados que provam que a solução proposta, a arquitectura ScalSec, além de não gerar mais tráfego é ainda mais rápida que a arquitectura PKIX.

O mesmo se verifica com o *SSID global*, o qual, não provoca um maior processamento por parte do *supplicant*, mantendo o tempo de autenticação igual.

De um modo geral a introdução desta arquitectura reduz a informação adicional necessária nos certificados, permitindo ainda que o tamanho dos certificados seja mais reduzido quando comparado com a arquitectura PKIX. Essa redução do tamanho faz com que se tirem dividendos quer no tempo de processamento dos certificados, bem como no tráfego médio gerado.

Capítulo 7

Conclusões

Os objectivos propostos foram integralmente atingidos.

Após a identificação dos problemas delineou-se uma solução, a qual foi divida em duas partes distintas permitindo desta forma responder de modo mais claro e preciso aos problemas identificados.

Assim para responder ao problema da correcta identificação do servidor de autenticação, foi elaborada e testada uma solução usando o *SSID global*, que consistiu na criação de regras que delimitam logicamente o SSID (normal) de um FQDN através de um carácter separador, sendo o servidor de autenticação identificado pelo FQDN. A limitação do SSID como identificador local é descrita em [6] de forma explícita, sendo também referenciada pela contribuição [11] a sua falta de estrutura lógica, questão respondida pela introdução do *SSID global*.

Relativamente ao problema da validação do servidor de autenticação utilizou-se a arquitectura ScalSec, introduzindo-se um novo modelo de certificação dos certificados do nó recorrendo ao DNSSEC. Este novo método tem a vantagem de poder certificar vários servidores de forma descentralizada, necessitando apenas que o certificado do nó tenha sido o mesmo a assinar os diferentes certificados do servidor de autenticação. A arquitectura ScalSec permite uma revogação dos certificados em tempo reduzido (tempo máximo de 2 TTLs) permitindo maior flexibilidade que a solução PKIX. Esta arquitectura é tão versátil que suporta vários tipos de URI, apesar de neste momento apenas ter sido testada e validada com dois tipos o FQDN e o IP.

Para a solução ScalSec foi utilizado o lwresd daemon. Este *resolver* ocupa pouca memória e o poder de processamento necessário é muito pouco para a maioria das estação portátis actuais.

Uma melhoria à solução ScalSec passa pela criação de uma extensão no EAP-TLS que permita de forma transparente obter um registo (do tipo H, K) deixando assim de haver necessidade de o processo de autenticação ser executado em duas fases distintas e poder passar a ser executado em apenas uma fase, ao estilo da extensão já definida em [7] para o OCSP. Devido ao facto da solução desenvolvida nao ter um método de autenticação nativo, torna-se necessário um pré-processo de autenticação no *supplicant*, não existe contudo perigo de roubo de credenciais, isto porque, as credenciais são transmitidas obrigatoriamente mediante protocolos que usam cifra, de acordo com

as novas recomendações do IETF, estando assim de acordo com as mais recentes boas práticas em [24].

No decorrer deste trabalho houve alguns contratempos, nomeadamente a compreensão do libiptc, bem como a pouca documentação dísponível da API da biblioteca lwresd, factos que originaram atrasos na implementação do trabalho não permitindo a realização do número ideal de testes, sendo certo que mais testes e cenários deveriam ter sido testados para a obtenção de resultados mais fidedignos e com uma menor margem de erro. No entanto os resultados obtidos permitem concluir que as soluções desenvolvidas têm potencial para contrariar a adopção do PKIX e serem consideradas alternativas válidas.

7.1 Contribuições Relevantes

As contribuições relevantes que foram desenvolvidas ao longo deste trabalho são as seguintes:

- Alteração do processo de autenticação para que este suporte a solução ScalSec, tornando o processo mais fléxivel que o PKIX bem como mais simples e barato.
- Introdução do *SSID global* que permite a identificação do servidor de autenticação baseando-se nas informações do SSID.

Em termos de trabalho desenvolvido as minhas contribuições foram as seguintes:

- Alteração ao *supplicant* para que este suporte os modelos ScalSec e PKIX de uma forma transparente e sem intervenção por parte do utilizador.
- Alteração ao *supplicant* para que este suporte o *SSID global* sem ser necessário o uso da ScalSec.
- Procedeu-se ainda a uma alteração do *supplicant* no modo de validação da redes Wi-Fi, que não permite o uso de protocolos de autenticação sem cifra, por exemplo PAP, aquando do uso do *SSID global*.
- Por último foi criada uma *framework* de gestão e administração que torna o processo de gestão de certificados e de registos DNS automático.

7.2 Trabalho Futuro

Nesta secção serão indicados os pontos que se consideram mais relevantes melhorar nesta implementação em termos de efeciência de processos, bem como na completa implementação dos diversos cenários de uso. O trabalho futuro será separado em dois grupos, o primeiro descreverá o trabalho a ser desenvolvido nos diversos ficheiros executáveis e o segundo grupo o trabalho a ser desenvolvido nos diferentes scripts.

Assim nos executáveis é necessário:

- Implementar no *supplicant* a indicação do estado quando se está a proceder a uma autenticação ScalSec que neste momento é inexistente.
- No módulo de firewall aquando da criação do backup das regras da firewall deverão ser suportadas as diversas tabelas existentes na estação, actualmente apenas suporta a tabela *filter*.
- A troca da biblioteca de lwresd em favor da ldns, que recentemente se tornou muito mais completa tendo maior compatibilidade para os diversos sistemas existentes, por ex: suporte de 64bits.
- Nas ferramentas criadas, mais propriamente na ferramenta 'getCAPKey' é recomendado também a conclusão do suporte das outras funções de Digest especificados, uma vez que neste momento apenas suporta SHA1.

Nos scritps de gestão criados são precisas as seguintes modificações:

- Dispôr da possibilidade de actualização dinâmica do servidor de DNS (DDNS), através do uso do 'nsupdate' e do 'rndc'.
- No script put_key deverá ser adicionada um novo switch que possibilite escolher o servidor remoto para onde enviar os registos.
- Conceber um processo de actualização das "zonas hierarquicamente superiores" que torne o *rollover* de uma chave KSK o mais transparente e rápida possível.
- Passar a totalidade desses scritps para uma linguagem mais poderosa e rápida, como é o caso do perl.
- Criar um utilizador para tratar do processo de actualização de registos e certificados, desta forma é criada a separação lógica da execução dos scritps *(sandbox)*, tornando mais díficil explorar possíveis falhas nos scritps ou na sua lógica, tornando ainda possível a imposição de restrição de acessos.

Bibliografia

[1] *Public key validation for the DNS security extensions*, volume I, 2001. English.

[2] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible authentication protocol (eap). RFC 3748 (Proposed Standard), Jun 2004.

[3] F. Adrangi, V. Lortz, F. Bari, and P. Eronen. Identity selection hints for the extensible authentication protocol (eap). RFC 4284 (Informational), Jan 2006.

[4] Márton Anka's. Ssl blacklist. Internet. http://www.codefromthe70s.org/sslblacklist-badcerts.aspx.

[5] Suranjith Ariyapperuma and Chris J. Mitchell. Security vulnerabilities in dns and dnssec. In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, pages 335–342, Washington, DC, USA, 2007. IEEE Computer Society.

[6] J. Arkko, B. Aboba, J. Korhonen, and F. Bari. Network discovery and selection problem. RFC 5113 (Informational), Jan 2008.

[7] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport layer security (tls) extensions. RFC 4366 (Proposed Standard), apr 2006.

[8] T. Dierks and C. Allen. The tls protocol version 1.0. RFC 2246 (Proposed Standard), jan 1999. Obsoleted by RFC 4346, updated by RFC 3546.

[9] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.1. RFC 4346 (Proposed Standard), apr 2006. Updated by RFCs 4366, 4680, 4681.

[10] R. Housley, W. Polk, W. Ford, and D. Solo. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 3280 (Proposed Standard), apr 2002. Updated by RFCs 4325, 4630.

[11] *Co-existence of Different Authentication Models*, volume IEEE Contribution 11-03-0827, Mar. 2003.

[12] S. Josefsson. Storing certificates in the domain name system (dns). RFC 4398 (Proposed Standard), Mar 2006.

[13] D. Kaminsky. It's the end of the cache as we know it. internet, July 2008. www.doxpara.com/DMK_BO2K8.ppt.

[14] O. Kolkman and R. Gieben. Dnssec operational practices. RFC 4641 (Informational), sep 2006.

[15] O. Kolkman, J. Schlyter, and E. Lewis. Domain name system key (dnskey) resource record (rr) secure entry point (sep) flag. RFC 3757 (Proposed Standard), Apr 2004. Obsoleted by RFCs 4033, 4034, 4035.

[16] Chris J. Mitchell. Attacking the dns protocol security paper v2. *Sainstitute*, 2003.

[17] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol - ocsp. RFC 2560 (Proposed Standard), Jun 1999.

[18] NIST. *Recommendation for Key Management – Part 2: Best Practices for Key Management Organizations*, volume I. NIST Special Publication 800-57, Apr 2005. English.

[19] Compatible Openpgp, Vlastimil Klíma, and Tomás Rosa. Attack on private signature keys of the openpgp format, pgp programs and other applications compatible with openpgp, 2002.

[20] C. Rigney. Radius accounting. RFC 2866 (Informational), jun 2000. Updated by RFCs 2867, 5080.

[21] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote authentication dial in user service (radius). RFC 2865 (Draft Standard), jun 2000. Updated by RFCs 2868, 3575, 5080.

[22] S. Venaas S. Winter, M. McCauley. Radsec version 2 - a secure and reliable transport for the radius. IETF proposed Draft, Feb. 2008.

[23] Anil Sagar. Dnssec: A vision. Presentation, Indian Computer Emergency Response Team (CERT-In), May 2007.

[24] D. Stanley, J. Walker, and B. Aboba. Extensible authentication protocol (eap) method requirements for wireless lans. RFC 4017 (Informational), Mar 2005.

[25] Lars Strand. 802.1 x port based authentication howto. internet, Aug. 2004.

Anexos A

Autenticação de redes Wi-Fi utilizando o EAP-PEAP

A estação dá inicio ao pedido de adesão à rede tentado para isso associar-se ao AP, ou APs, que anunciam o SSID, pré-configurado no *supplicant*. O pedido é recusado por parte do AP, apontando como meio de autenticação o EAP; pode, no entanto, ser aceite o pedido de associação, caso a associação tenha sido feita à pouco tempo e a ligação por qualquer motivo tenha caído.

De seguida, o *supplicant* faz um pedido ao AP para iniciar uma autenticação EAP, ao qual o AP responde com um pergunta, com que identidade a estação se vai autenticar.

Perante esta pergunta de autenticação, o *supplicant*, procederá em concordância aos métodos pré-configurados. Neste caso, será explicado uma autenticação utilizando o protocolo EAP-PEAP v0 que utilizará durante na segunda fase de autenticação o MSChapv2; este método servirá de base para qualquer autenticação que utilize um *supplicant* e o EAP com TLS.

Este método de autenticação, cria um túnel seguro (TLS) entre o cliente e o servidor de autenticação, permitindo uma troca de certificados X.509, que após verificação por ambas as partes, permite autenticar o servidor/cliente e autorizar o acesso à rede.

Os certificados X.509, usados na autenticação, serão no mínimo 2, o certificado de CA e o certificado do servidor de autenticação, sendo que para serem considerados válidos, a norma X.509 especifica que necessitam de ser aprovados segundo o "Certification path validation algorithm".

O formato da troca de mensagens usado pelo EAP já foi abordado no Estado da Arte na secção 3.3, aqui será focada a negociação da ligação TLS que, neste caso, representa a fase EAP-Request-Challenge/EAP-Response-Challenge:

- O cliente começa por gerar uma chave aleatória, que envia para o servidor, juntamente com uma mensagem de Hello.
- Por seu lado o servidor responde com uma mensagem de Hello e um número aleatório gerado por este.
- Aí o servidor e cliente combinam, as cifras, a compressão e outros parâmetros
- O servidor de seguida, envia o certificado de CA e o seu certificado
- O cliente verifica a validade dos certificados

- O cliente de seguida gera um Pré Master Key, baseando-se num número aleatório e envia a Pré Master Key para o servidor cifrada usando como chave, a chave pública do certificado do servidor.
- Ambos os intervenientes gerarão uma Master key igual baseando-se nas outras 3 chaves já trocadas
- O cliente deverá aí alterar a sua chave de cifra da ligação para a chave Master Key e anunciar o fim do handshake, o mesmo acontecerá do lado do servidor.

Pode ser visto o esquema de autenticação TLS, no qual apenas o servidor é autenticado (PEAP) na figura 7.1.

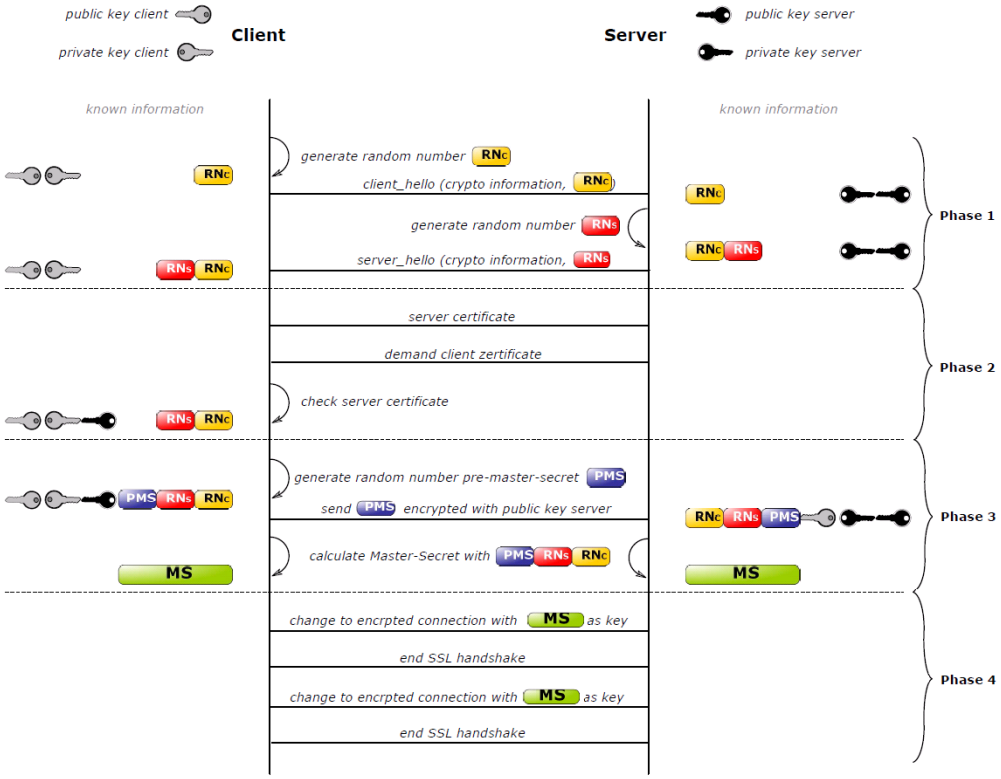


Figura 7.1: Autenticação EAP-PEAP

Após o estabelecimento desse túnel seguro, dá-se por terminada a primeira fase de autenticação. O *supplicant* irá utilizar o túnel para a passagem de mensagens EAP-PEAP com o servidor de autenticação. Dá-se assim inicio à segunda fase de autenticação, as mensagens trocadas terão a seguinte ordem:

- Será lançado um pedido de identificação ao *supplicant*, por parte do servidor de autenticação
- Supplicant responde com a sua identidade
- O servidor de autenticação requer uma ligação PEAP v0 - MSChapv2, com um desafio.

- Supplicant responde ao desafio, cifrando os dados com a resposta
- Se bem sucedida a resposta, o servidor de autenticação responde ao desafio com uma resposta de sucesso
- O *supplicant* responde, ao servidor de autenticação, também com uma resposta de sucesso, dando por terminada a autenticação

Depois de terminada a autenticação PEAP, se tiver sido correcta, o *supplicant* fechará a sessão de TLS com o cliente, se não tiver sido correcta, irá tentar novamente se autenticar com o MSChapv2.

Depois de fechado o túnel o servidor de autenticação irá dar ordens ao AP (NAS) para que deixe a estação se associar à rede.

Anexos B

Certificados

Certificado servidor

```
Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=PT, ST=Porto, O=INESC, OU=INESC Porto, CN=ScalSec/emailAddress=hos
    Validity
        Not Before: Mai 29 18:33:28 2008 GMT
        Not After : Mai 29 18:33:28 2009 GMT
    Subject: C=PT, ST=Porto, O=INESC, OU=INESC Porto, CN=ScalSec/emailAddress=ho
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        RSA Public Key: (512 bit)
            Modulus (512 bit):
                00:c3:e9:07:09:87:a7:dc:4d:b6:c6:5f:25:85:6e:
                1f:21:a7:55:44:f9:8a:bd:00:2e:ef:6d:12:df:00:
                92:2e:ca:57:eb:00:f6:83:15:46:2b:7e:57:52:eb:
                11:ee:34:73:7d:0e:d5:b9:5a:98:56:4a:41:9c:fc:
                63:f9:0f:3a:bd
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Key Identifier:
            56:E4:77:58:6D:2E:7F:54:22:27:74:B1:95:1E:9F:EE:AA:91:5C:F1
        X509v3 Authority Key Identifier:
            keyid:56:E4:77:58:6D:2E:7F:54:22:27:74:B1:95:1E:9F:EE:AA:91:5C:F1
            DirName:/C=PT/ST=Porto/O=INESC/OU=INESC Porto/CN=ScalSec/emailAddres
            serial:01

        X509v3 Basic Constraints:
            CA:TRUE
        Netscape Cert Type:
            SSL CA
        X509v3 Subject Alternative Name:
```

```
            DNS:radius.porto.pt
    Signature Algorithm: sha1WithRSAEncryption
        22:ec:e1:d9:9a:ca:2d:59:8d:2f:bf:7c:0e:cb:b6:e3:f2:b5:
        4d:c8:f5:46:af:77:cd:c0:82:33:cc:04:bf:9f:53:a5:be:e7:
        bc:26:98:b9:d8:f5:20:d0:22:fb:21:cc:67:5a:70:b9:ce:be:
        c4:76:ee:da:16:02:9f:c7:89:42
```

Certificado do servidor

```
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=PT, ST=Porto, O=INESC, OU=INESC Porto, CN=ScalSec/emailAddress=hostmast
        Validity
            Not Before: Jun 29 18:33:28 2008 GMT
            Not After : Jul 30 18:33:28 2008 GMT
        Subject: C=PT, ST=Porto, O=INESC, OU=INESC Porto - Unidade UTM, CN=ScalSec/emailA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (384 bit)
                Modulus (384 bit):
                    00:9c:44:7d:cb:c0:87:f0:e8:47:77:70:03:12:07:
                    ee:18:5c:76:28:a2:23:6f:c4:59:ed:d1:d8:3d:7a:
                    37:c6:17:c1:71:32:e6:b7:af:6b:63:61:64:5a:4c:
                    72:09:5d:d7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Server, Object Signing
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Key Identifier:
                8D:95:9A:44:76:C8:42:C0:AE:80:18:AF:A5:00:46:2C:5B:02:CC:FD
            X509v3 Authority Key Identifier:
                keyid:56:E4:77:58:6D:2E:7F:54:22:27:74:B1:95:1E:9F:EE:AA:91:5C:F1

            X509v3 Subject Alternative Name:
                DNS:radius
    Signature Algorithm: sha1WithRSAEncryption
        9f:2b:f2:ad:e2:11:0e:23:66:67:34:ad:84:f9:7d:de:c7:df:
        03:7c:e1:4c:30:6e:fc:c6:73:15:4f:fd:7b:4c:d2:3f:1b:79:
        4b:2e:3f:0e:d7:99:e4:13:7d:75:23:2e:6a:20:73:03:9b:b6:
```

```
1b:58:9e:54:aa:35:b1:5d:d9:9c
```

generate_certs - Script de geração de certificados

```
#! /bin/bash

USER=`whoami`
PWD=`pwd`
: ${DEBUG:=0}
SUDO_NEEDED=0

#GEN CA Request
PASS="testing"
CAKEY="private/cacert.key"
CACERT="cacert.pem"
CATOP="./CA"

#Used for execute a command and prints the command used if DEBUG mode is set
execute() {
if [ ${DEBUG} -eq 1 ]; then
echo "$@"
fi
#Check if commands are executed in root or sudo mode
if [ ${SUDO_NEEDED} -eq 1 ]; then
sudo $@
else
$@
fi

}

help()
{
 echo "Usage:"
 echo "generate_certs  <options>"
 echo "all - CA cert and server cert"
 echo "ca - Just the CA cert"
 echo "server - Just the server cert"
 echo "Other variable options:"
 echo " \$DEBUG - Enable Debug!"
}

gen_CA()
{
#Generate a CA
```

```
CONFIG="-config ca.cnf -batch "
REQ="openssl req $CONFIG"
DAYS="-days 365"
    ## Assume the rest of them doesn't exis
    if [ ! -d ${CATOP}/certs ] || [ ! -d ${CATOP}/private ]; then
# create the directory hierarchy
        execute mkdir ${CATOP}
        execute mkdir ${CATOP}/certs
        execute mkdir ${CATOP}/crl
        execute mkdir ${CATOP}/newcerts
        execute mkdir ${CATOP}/private
        execute chmod 600 ${CATOP}/private
if [ ! -f ${CATOP}/serial ]; then
         echo "01" | execute tee -a ${CATOP}/serial > /dev/null 2>&1
execute touch ${CATOP}/index.txt
fi
    fi
    #Root protected dir
    execute test -f ${CATOP}/${CAKEY}
    if [ $? -eq 1 ]; then
    #Generating Key
echo "Making CA certificate ...and key"
execute $REQ -new ${CONFIG} -passin pass:${PASS} -passout pass:${PASS} \
  -keyout ${CATOP}/${CAKEY} -out careq.csr $DAYS
execute openssl ca ${CONFIG} -passin pass:${PASS} -out ${CATOP}/${CACERT} \
  -keyfile ${CATOP}/${CAKEY} -selfsign -extensions v3_ca -in careq.csr
    else
     execute openssl ca ${CONFIG} -passin pass:${PASS} -out ${CATOP}/${CACERT} \
  -keyfile ${CATOP}/${CAKEY} -selfsign -extensions v3_ca -in careq.csr
    fi
    if [ ! -z ${ZONE} ]; then
#Recriate RADIUS server Certificates
put_dnskey ${ZONE} > /dev/null 2>&1
   fi
}


clean()
{
 execute rm *.pem >& /dev/null
 execute rm *.csr >& /dev/null
 execute rm *.key >& /dev/null
 execute rm -r ${CATOP} >& /dev/null
}
```

```
gen_ServerCerts()
{
#Server Certificates
SVKEY="servcert.key"
SVCERT="servcert.pem"
CONFIG="-config server.cnf -batch "
REQ="openssl req $CONFIG"
DAYS="-days 31"

#check careq.csr and cacert and private key
execute test -f "${CATOP}/${CAKEY}"
if [ $? -eq 1 ]; then
echo "Ignored: Ca private key non existant!"
fi

if [ ! -f ${CATOP}/${CACERT} ]; then
echo "Ca certificate non existant!"
#No CA cert found, generating one
gen_CA
fi

echo "Generate Certificate Signing Request ..."
#Generate Certificate Signing Request
execute $REQ -new -passin pass:${PASS} -passout pass:${PASS} \
 -keyout $SVKEY -out srvreq.csr $DAYS
echo "Generate server CA assined Certificate"
execute openssl ca ${CONFIG} -passin pass:${PASS} -in srvreq.csr \
-out $SVCERT $DAYS
echo "Generate X509 Certificate"
execute openssl x509 -extfile server.cnf -in $SVCERT -outform PEM \
-out PKeySCertificate.pem
}


#No need for
#Gen Client Cert
#CLIENTKEY="certclient.key"
#CLIENTCERT="certclient.pem"
#echo "Generate Client Key and Certificate Signing request ..."
#Generate Certificate Signing Request
#execute $REQ -new -passin pass:${PASS} -passout pass:${PASS} -keyout $CLIENTKEY -ou

#echo "Generate server CA assined Certificate"
#execute openssl ca -batch -extensions usr_cert -passin pass:testing ${CONFIG} -in c
# -out $CLIENTCERT $DAYS
```

```
#echo "Generate X509 Certificate"
#execute openssl x509 -extfile openssl.cnf -in $CLIENTCERT -outform PEM -out PKeyCCertifi

if [ ! $# -eq 1 ]; then
 echo "Please especify what certs want to generate"
 help
 exit 1
fi

if [ "$1" = "--help" ]; then
help
exit 0
fi

#Check Root
if [ ! "${USER}" = "root" ]; then
echo "You are not root user, using sudo!"
echo "Check if you are in sudoers file!"
SUDO_NEEDED=1
fi

if [ ! -z $2 ]; then #assume it's a zone
ZONE="$2"
fi

case "${1}" in
"all") clean; gen_CA; gen_ServerCerts ;;
"ca") clean; gen_CA ;;
"server") gen_ServerCerts ;;
"clean") clean;;
*) echo "Unknown option"; exit 1;;
esac


[ ca ]
default_ca = CA_default # The default ca section

[ CA_default ]
dir = ./CA # Where everything is kept
certs = $dir/certs # Where the issued certs are kept
crl_dir = $dir/crl # Where the issued crl are kept
database = $dir/index.txt # database index file.
#unique_subject = no # Set to 'no' to allow creation of
# several ctificates with same subject.
new_certs_dir = $dir/newcerts # default place for new certs.
```

```
certificate = $dir/cacert.pem  # The CA certificate
serial = $dir/serial  # The current serial number
crlnumber = $dir/crlnumber # the current crl number
# must be commented out to leave a V1 CRL
crl = $dir/crl.pem  # The current CRL
private_key = $dir/private/cacert.key # The private key
RANDFILE = $dir/private/.rand # private random number file

x509_extensions = v3_ca # The extentions to add to the cert
name_opt  = ca_default # Subject Name options
cert_opt  = ca_default # Certificate field options

default_days = 365 # how long to certify for
default_crl_days= 30 # how long before next CRL
default_md = sha1 # which md to use.
preserve = no # keep passed DN ordering

policy = policy_match

# For the CA policy
[ policy_match ]
countryName = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

[ policy_anything ]
countryName = optional
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

####################################################################
[ req ]
default_bits = 512
default_keyfile  = servcert.key
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = v3_req # The extentions to add to the self signed cert
string_mask = nombstr
```

```
req_extensions = v3_req # The extensions to add to a certificate request, necessary for a

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = PT
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Porto

localityName = Locality Name (eg, city)
localityName_default = Porto

0.organizationName = Organization Name (eg, company)
0.organizationName_default = INESC

organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = INESC Porto

commonName = Common Name (eg, YOUR name)
commonName_default = ScalSec
commonName_max = 64

emailAddress = Email Address
emailAddress_default = hostmaster@porto.pt
emailAddress_max = 64

SET-ex3 = SET extension number 3

[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20
challengePassword_value = #

unstructuredName = An optional company name

[ usr_cert ]
basicConstraints=CA:FALSE
nsCertType = server, objsign
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName=DNS:radius

[ v3_req ]
```

```
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
basicConstraints = CA:TRUE
nsCertType = sslCA

[ crl_ext ]
authorityKeyIdentifier=keyid:always,issuer:always

[ proxy_cert_ext ]
basicConstraints=CA:TRUE
nsCertType = server
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

[ ca ]
default_ca = CA_default # The default ca section

[ CA_default ]
dir = ./CA # Where everything is kept
certs = $dir/certs # Where the issued certs are kept
crl_dir = $dir/crl # Where the issued crl are kept
database = $dir/index.txt # database index file.
#unique_subject = no # Set to 'no' to allow creation of
# several ctificates with same subject.
new_certs_dir = $dir/newcerts # default place for new certs.

certificate = $dir/cacert.pem  # The CA certificate
serial = $dir/serial  # The current serial number
crlnumber = $dir/crlnumber # the current crl number
# must be commented out to leave a V1 CRL
crl = $dir/crl.pem  # The current CRL
private_key = $dir/private/cacert.key # The private key
RANDFILE = $dir/private/.rand # private random number file

x509_extensions = usr_cert # The extentions to add to the cert
name_opt  = ca_default # Subject Name options
cert_opt  = ca_default # Certificate field options
```

```
default_days = 365 # how long to certify for
default_crl_days= 30 # how long before next CRL
default_md = sha1 # which md to use.
preserve = no # keep passed DN ordering

policy = policy_match

# For the CA policy
[ policy_match ]
countryName = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

[ policy_anything ]
countryName = optional
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

####################################################################
[ req ]
default_bits = 384
default_keyfile  = servcert.key
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = v3_ca # The extentions to add to the self signed cert
string_mask = nombstr
req_extensions = v3_req # The extensions to add to a certificate request, necessary for a

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = PT
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Porto

localityName = Locality Name (eg, city)
```

```
localityName_default = Porto

0.organizationName = Organization Name (eg, company)
0.organizationName_default = INESC

organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = INESC Porto - Unidade UTM

commonName = Common Name (eg, YOUR name)
commonName_default = ScalSec
commonName_max = 64

emailAddress = Email Address
emailAddress_default = hostmaster@porto.pt
emailAddress_max = 64

SET-ex3 = SET extension number 3

[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20
challengePassword_value = #

unstructuredName = An optional company name

[ usr_cert ]
basicConstraints=CA:FALSE
nsCertType = server, objsign
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName=DNS:radius

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
basicConstraints = CA:TRUE
nsCertType = sslCA

[ crl_ext ]
authorityKeyIdentifier=keyid:always,issuer:always
```

```
[ proxy_cert_ext ]
basicConstraints=CA:TRUE
nsCertType = server
nsComment = "OpenSSL Generated Certificate"
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo
```

Anexos C

DNS

gen_key - Script de geração de chaves

```
#! /bin/bash

USER=`whoami`
PWD=`pwd`
: ${DEBUG:=0}
SUDO_NEEDED=0

BINDDIR="/etc/bind/"

ALGTYPE="RSASHA1"
DEPTH=512

help()
{
 echo "Usage:"
 echo "gen_key <zone name> <key type> -a <algorithm type> -d <depth bits>"
 echo "key type - specifies if the key is of type: KSK | ZSK"
 echo "algorithm type - can be of the followinf types"
echo "RSA | RSAMD5 | DH | DSA | RSASHA1 | HMAC-MD5 | HMAC-SHA1 | HMAC-SHA224 | HMAC-
 echo "depth bits - bits depth of signing key [512...4096]"
 echo "Other variable options:"
 echo " \$DEBUG - Enable Debug!"
}

#Used for execute a command and prints the command used if DEBUG mode is set
execute() {
if [ ${DEBUG} -eq 1 ]; then
echo "$@"
fi
#Check if commands are executed in root or sudo mode
if [ ${SUDO_NEEDED} -eq 1 ]; then
sudo $@
```

```
else
$@
fi

}

if [ "$1" = "--help" ]; then
help
exit 0
fi

if [ $# -lt 2 ]; then
 echo "Please especify all needed parameters"
 help
 exit 1
fi

# add our domain
DOMAIN=$1;

case "$2" in
"KSK")
EXTRAOPTS="-f KSK";;
"ZSK")
EXTRAOPTS="";;
*)  echo "Unknown Key Type"; exit 1;;
esac

until [ -z "$3" ]; do
        case "$3" in
          "-a")
              ALGTYPE="$4";;
          "-d")
              DEPTH="$4";;
          *)  ;; #Ignore
        esac
        shift
done

#Check Root
if [ ! "${USER}" = "root" ]; then
echo "You are not root user, using sudo!"
echo "Check if you are in sudoers file!"
SUDO_NEEDED=1
fi
```

```
echo "Generating a keyset!"

if [ ! -d ${BINDDIR}/${DOMAIN} ]; then
echo "Error: Zone dir doesn't exist"
exit 1
fi

cd ${BINDDIR}/${DOMAIN}

if [ "${EXTRAOPTS}" = "" ]; then
if [ ! -e "zsk" ]; then
mkdir zsk > /dev/null 2>&1
fi
cd zsk > /dev/null 2>&1
else
if [ ! -e "ksk" ]; then
mkdir ksk > /dev/null 2>&1
fi
cd ksk > /dev/null 2>&1
fi


execute dnssec-keygen -r /dev/urandom -a ${ALGTYPE} -b ${DEPTH} -n ZONE ${EXTRAOPTS}


if [ $? -eq 0 ]; then
echo "Sucess: Zone key sucessfully created"
fi

cd ${PWD}
```

dnsupdate - Script de instalação da framework

```
#!/bin/bash

USER=`whoami`
: ${DEBUG:=0}
SUDO_NEEDED=0

dnsupdate_dir="/etc/dnsupdate"
dnsupdate_daily="${dnsupdate_dir}/dnsupdate-daily"
dnsupdate_monthly="${dnsupdate_dir}/dnsupdate-monthly"
tasks_file="${dnsupdate_dir}/zones"
```

```
sign_zone="/etc/bind/sign_zone"
gen_key="/etc/bind/gen_key"

#Used for execute a command and prints the command used if DEBUG mode is set
execute() {
if [ ${DEBUG} -eq 1 ]; then
echo "$@"
fi
#Check if commands are executed in root or sudo mode
if [ ${SUDO_NEEDED} -eq 1 ]; then
sudo $@
else
$@
fi
}

help()
{
 echo "Usage:"
 echo "dnsupdate <dns zone>"
 echo "Specifies to wich dns zone are the maintenance tasks"
 echo "It criates rules and tasks for a first time use"
 echo "It also supports multi zone maintenance"
 echo "Other variable options:"
 echo " \$DEBUG - Enable Debug!"
}

daily()
{
if [ ! "$2" = "" ]; then
TIME="$2"
else
#This parameter is optional...
TIME=""
fi

cat "${tasks_file}" | while read ZONE
do
execute ${dnsupdate_daily} ${ZONE} ${TIME}
done
}

monthly()
{
if [ ! "$2" = "" ]; then
```

```
TASKS="$2"
else
echo "No tasks given"
exit 1
fi

cat "${tasks_file}" | while read ZONE
do
execute ${dnsupdate_monthly} ${ZONE} ${TASKS}
done
}

first_usetasks()
{
if [ ! -e ${dnsupdate_dir} ]; then
execute mkdir ${dnsupdate_dir}
fi

if [ ! -e ${dnsupdate_dir}/dnsupdate ]; then
execute cp dnsupdate ${dnsupdate_dir}
#Check right permissions
execute chmod +x ${dnsupdate_dir}/dnsupdate
fi

if [ ! -e ${dnsupdate_dir}/dnsupdate-daily ]; then
execute cp dnsupdate-daily ${dnsupdate_dir}
#Check right permissions
execute chmod +x ${dnsupdate_dir}/dnsupdate-daily
fi
if [ ! -e ${dnsupdate_dir}/dnsupdate-monthly ]; then
execute cp dnsupdate-monthly ${dnsupdate_dir}
#Check right permissions
execute chmod +x ${dnsupdate_dir}/dnsupdate-monthly
fi

if [ ! -e ${dnsupdate_dir}/nsupdater ]; then
execute cp nsupdater ${dnsupdate_dir}
#Check right permissions
execute chmod +x ${dnsupdate_dir}/nsupdater
fi

if [ ! -e ${sign_zone} ]; then
execute cp sign_zone ${sign_zone}
#Check right permissions
execute chmod +x ${sign_zone}
```

```
fi

if [ ! -e ${gen_key} ]; then
execute cp gen_key ${gen_key}
#Check right permissions
execute chmod +x ${gen_key}
fi

if [ ! -e ${tasks_file} ]; then
execute touch ${tasks_file}
fi
}

if [ $# -lt 1 ]; then
 echo "Please especify to wich zone the tasks are met"
 help
 exit 1
fi

if [ "$1" = "--help" ]; then
help
exit 0
fi

#Check Root
if [ ! "${USER}" = "root" ]; then
echo "You are not root user, using sudo!"
echo "Check if you are in sudoers file!"
SUDO_NEEDED=1
fi

case "$1" in
"daily") daily $@; exit 0;;
"monthly") monthly $@; exit 0;;
*) ZONE="$1";;
esac

first_usetasks

#Check if zone file is in tasks_file
execute cat ${tasks_file} | grep "${ZONE}" > /dev/null 2>&1
RES=$?

if [ ${RES} -eq 1 ]; then
echo "$ZONE" | execute tee -a ${tasks_file} > /dev/null 2>&1
```

```
#Update Crontab tasks file
execute ${dnsupdate_daily} ${ZONE} 24h > /dev/null 2>&1
execute ${dnsupdate_monthly} ${ZONE} 0 > /dev/null 2>&1
else
echo "Zone is already in database!"
fi
```

dnsupdate_daily - Script de assinatura diária das zonas

```
#!/bin/bash

BINDDIR="/etc/bind/"

#Service names
BIND_SERVICE="bind9"

#Expiration time of rrsigs in seconds
HOUR12="43200"
HOUR24="86400"
HOUR36="129600"

PWD=`pwd`

sign_zone="/etc/bind/sign_zone"

SIGTIME=$HOUR36

daily_tasks()
{
#Keys location
KSKDIR="$BINDDIR/${ZONE}/ksk/"
ZSKDIR="$BINDDIR/${ZONE}/zsk/"

#oldest key signs
KSKKEY="`ls -1 -c -r ${KSKDIR}/K*.key`"
ZSKKEY="`ls -1 -c -r ${ZSKDIR}/K*.key`"


#Regenerate DNS RRSIGs
${sign_zone} ${ZONE} -k ${KSKKEY} -z ${ZSKKEY} -e ${SIGTIME} # > /dev/null 2>&1

#Reload DNS Server
rndc reload # ${ZONE} > /dev/null 2>&1
```

```
}

if [ $# -lt 1 ]; then
 echo "Please especify to what zone the task are destined!"
 exit 1
fi

ZONE="$1"

if [ ! -z $2 ] && [ "$2" = "24h" ]; then #assume it's sig time
SIGTIME=$HOUR24
elif [ ! -z $2 ] && [ "$2" = "12h" ]; then
SIGTIME=$HOUR12
fi

daily_tasks
```

dnsupdate_monthly - Script de assinatura mensal das zonas

```
#!/bin/bash

DATE=`date +%x`
DATEDAY=`date +%D | awk -F'/' '{print $2}'`
DATEMON=`date +%D | awk -F'/' '{print $1}'`

TIMEMINUTC=`date -u +%R | awk -F':' '{print $2}'`
TIMEHOURUTC=`date -u +%R | awk -F':' '{print $1}'`

TIMEMIN=`date +%R | awk -F':' '{print $2}'`
TIMEHOUR=`date +%R | awk -F':' '{print $1}'`

HOURFIX=02
MINFIX=10

UPDATE_FILE="/tmp/.crontabupdate"
TTL=6 #(hours)

dnsupdate_dir="/etc/dnsupdate"
dnsupdate_daily="${dnsupdate_dir}/dnsupdate daily"
dnsupdate_monthly="${dnsupdate_dir}/dnsupdate monthly"
nsupdater="${dnsupdate_dir}/nsupdater"

gen_key="/etc/bind/gen_key"
```

```
debug()
{
echo "SET: | ${TIMEMIN} ${TIMEHOUR} | ${DATEDAY} ${DATEMON} |"
echo "USE: | ${MINTOUSE} ${HOURTOUSE} | ${DAYTOUSE} ${MONTHTOUSE} |"
}

validate_day()
{
#Specify how much days each month have
MON31=( "1" "3" "5" "7" "8" "10" "12" )
MON30=( "4" "6" "9" "11" "0" "0" )
MON28=( "2" "0" "0" "0" "0" "0" )
#Arguments
DAY=$1
MONTH="$2"

#optimization 28 days rule
if [ ${DAY} -le 28 ]; then
return ${DAY};
fi

for i in `seq 0 6`;
do
if [ "${MON31[$i]}" = "${MONTH}" ]; then
return ${DAY};
fi

if [ "${MON30[$i]}" = "${MONTH}" ]; then
if [${DAY} -le 30 ]; then
return ${DAY};
else #Normalized day to 30
return 30;
fi
fi

if [ "${MON28[$i]}" = "${MONTH}" ]; then
if [${DAY} -gt 28 ]; then
#Normalized day to 28
return 28;
fi
fi
done

echo "Don't know what happened here!"
```

```
return 1;
}

getdays_month()
{
#Specify how much days each month have
MON31=( "1" "3" "5" "7" "8" "10" "12" )
MON30=( "4" "6" "9" "11" "0" "0" )
MON28=( "2" "0" "0" "0" "0" "0" )
#Arguments
MONTH="$1"

for i in `seq 0 6`;
do
if [ "${MON31[$i]}" = "${MONTH}" ]; then
return 31;
fi

if [ "${MON30[$i]}" = "${MONTH}" ]; then
return 30;
fi

if [ "${MON28[$i]}" = "${MONTH}" ]; then
return 28;
fi
done

echo "Don't know what happened here!"
return 28;
}

validate_adding()
{

if [ $HOURTEMP -ge 24 ]; then
let "DAYTEMP = DATEDAY + 1" # 1 day
let "HOURTOUSE = HOURTEMP - 24" # 24h time

#Check if day is valid for that month, because of february 28 days
validate_day ${DAYTEMP} ${DATEMON}
DAYTOUSE=$?

if [ ! "${DAYTEMP}" = "${DAYTOUSE}" ]; then
DAYTOUSE=1; #begining of month
#Increase a month
```

```
let "MONTHTOUSE = DATEMON - 1" # 1 month
if [ ${MONTHTOUSE} -gt 12 ]; then
#Revert a year in months
let "MONTHTOUSE -= 12"
fi
fi
else
HOURTOUSE=${HOURTEMP}
fi
}

validate_subtracting()
{

if [ $HOURTEMP -lt 0 ]; then
let "DAYTOUSE = DATEDAY - 1" # 1 day
let "HOURTOUSE = HOURTEMP + 24" # 24h time
if [ ${DAYTOUSE} -le 0 ]; then
#Reverte a month
let "MONTHTOUSE = DATEMON - 1" # 1 month
if [ ${MONTHTOUSE} -le 0 ]; then
#Revert a year
MONTHTOUSE=12;
fi
#Get last day of that month
getdays_month ${MONTHTOUSE}
DAYTOUSE=$?
fi
else
HOURTOUSE=${HOURTEMP}
fi
}

sync_time()
{
#Set default values
DAYTOUSE=${DATEDAY}
MONTHTOUSE=${DATEMON}
MINTOUSE=${TIMEMIN}
HOURTOUSE=${TIMEHOUR}

#Tasks to be executed at a fixed hours
DIFFMIN=0
DIFFHOUR=0
CARRY=0
```

```
#Calculate time diferences
let "DIFFMIN = TIMEMIN - TIMEMINUTC"
let "DIFFHOUR = TIMEHOUR - TIMEHOURUTC"

#Fixable update hour to 02h10
TIMEMIN=${MINFIX}
let "MINTOUSE = TIMEMIN + DIFFMIN"
if [ $MINTOUSE -lt 0 ]; then
let "MINTOUSE += 60"
CARRY="-1"
elif [ $MINTOUSE -gt 59 ]; then
let "MINTOUSE -= 60"
CARRY="1"
fi
TIMEHOUR=${HOURFIX};
let "HOURTEMP = TIMEHOUR + DIFFHOUR + CARRY"
if [ $DIFFHOUR -lt 0 ]; then
echo "local hour is less $DIFFHOUR hours than UTC time"
validate_subtracting
elif [ $DIFFHOUR -eq 0 ]; then
echo "local hour is the same as UTC time"
else
echo "local hour is more $DIFFHOUR hours than UTC time"
validate_adding
fi

#Use new dates
DATEDAY=${DAYTOUSE}
DATEMON=${MONTHTOUSE}
TIMEMIN=${MINTOUSE}
TIMEHOUR=${HOURTOUSE}
}

add_ttl()
{

#Add 1 TTL from Day Time
let "HOURTEMP = TIMEHOUR + TTL" # 24h + 1 TTL each time

validate_adding

}

addmonth()
{
```

```
#Actualiza o mês para o proximo mes
let "MONTHTOUSE = DATEMON + 1" # 12 months plus one

#Normalized month
if [ ${MONTHTOUSE} -gt 12 ]; then
let "MONTHTOUSE -= 12" # 12 months
fi

#Check if day is valid for that month, because of february 28 days
validate_day ${DATEDAY} ${MONTHTOUSE}
DAYTOUSE=$?

}

remove_2ttl()
{
#Remove 2 TTL from Day Time
let "HOURTEMP = TIMEHOUR - (TTL + TTL)" # 24h - 2 TTL each time

validate_subtracting
}


monthly_tasks()
{

case "${TASK}" in
#Add Crontab rule
"0")
#It's first time interation, next interation will be in 2 TTLS
addmonth
remove_2ttl

#Update Crontab tasks file
echo "SHELL=/bin/bash" >& ${UPDATE_FILE}
echo "PATH=${dnsupdate_dir}:${PATH}" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "${TIMEMIN} ${HOURTOUSE} ${DAYTOUSE} ${MONTHTOUSE} * ${dnsupdate_monthly} 1 > /
echo "${TIMEMIN} ${HOURTOUSE} * * * ${dnsupdate_daily} > /dev/null 2>&1" >> ${UPDATE
crontab ${UPDATE_FILE} > /dev/null 2>&1
unlink ${UPDATE_FILE};;

#Executado 2 TTL's antes
"1")
```

```
add_ttl

#Recriate DNS ZSK Key
${gen_key} ${ZONE} ZSK -b 1024 > /dev/null 2>&1
#Add new keys to zone
binddsupdater bind
#Add new key to register, so that a rollover is transparent to end user
${nsupdater} ${ZONE} H new add > /dev/null 2>&1
${nsupdater} ${ZONE} ZSK new add > /dev/null 2>&1
# Resign zones with new keys
${dnsupdate_daily} ${ZONE} 12h > /dev/null 2>&1
#edit Crontab with new file
echo "SHELL=/bin/bash" >& ${UPDATE_FILE}
echo "PATH=${dnsupdate_dir}:${PATH}" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "${TIMEMIN} ${HOURTOUSE} ${DAYTOUSE} ${MONTHTOUSE} * ${dnsupdate_monthly} 2 > /dev/n
crontab ${UPDATE_FILE} > /dev/null 2>&1
unlink ${UPDATE_FILE};;

#Executado 1 TTL's antes
"2")
#Set next time interation
add_ttl

#Regenerate DNS RRSIGs with new key
${nsupdater} ${ZONE} ZSK old change # set the old key as secondary key, no signing with i
${dnsupdate_daily} ${ZONE} 12h > /dev/null 2>&1
${nsupdater} ${ZONE} ZSK old change # reset changes made, set the predifined order
#edit Crontab with new file
echo "SHELL=/bin/bash" >& ${UPDATE_FILE}
echo "PATH=${dnsupdate_dir}:${PATH}" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "${TIMEMIN} ${HOURTOUSE} ${DAYTOUSE} ${MONTHTOUSE} * ${dnsupdate_monthly} 3 > /dev/n
crontab ${UPDATE_FILE} > /dev/null 2>&1
unlink ${UPDATE_FILE};;

#Executado on scheldule
"3")
#Set next time interation, will be one month less 2 TTLS
addmonth
remove_2ttl

#Remove old key from register, so that a rollover is transparent to end user
```

```
${nsupdater} ${ZONE} ZSK old del #> /dev/null 2>&1
${nsupdater} ${ZONE} H old del
#Resign zone
${dnsupdate_daily} ${ZONE} 24h > /dev/null 2>&1
#Update Crontab tasks file
echo "SHELL=/bin/bash" >& ${UPDATE_FILE}
echo "PATH=${dnsupdate_dir}:${PATH}" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "${TIMEMIN} ${HOURTOUSE} * * * ${dnsupdate_daily} > /dev/null 2>&1" >> ${UPDATE
echo "${TIMEMIN} ${HOURTOUSE} ${DAYTOUSE} ${MONTHTOUSE} * ${dnsupdate_monthly} 1 > /
crontab ${UPDATE_FILE} > /dev/null 2>&1
unlink ${UPDATE_FILE};;

*) echo "Unknown fase!"; exit 1;;
esac
}

if [ ! $# -eq 2 ]; then
 echo "Please especify what zone and monthly task should be performed"
 exit 1
fi

ZONE=$1
TASK=$2

sync_time

monthly_tasks
```

nsupdater - Script de actualização das zonas

```
#!/bin/bash

OLDEXT="old"
EXT="ds"
NEWEXT="new"

RRTYPE="TYPE120"
BINDDIR="/etc/bind"
ZONEPREFIX="db."

USER=`whoami`
: ${DEBUG:=0}
SUDO_NEEDED=0
```

```
PWD=`pwd`

#Used for execute a command and prints the command used if DEBUG mode is set
execute() {
if [ ${DEBUG} -eq 1 ]; then
echo "$@"
fi
#Check if commands are executed in root or sudo mode
if [ ${SUDO_NEEDED} -eq 1 ]; then
sudo "$@"
else
"$@"
fi
}

help()
{
 echo "Usage:"
 echo "nsupdater <zone> <key type> <status> <options>"
 echo "Zone - Dns zone to be updated"
 echo "Key type - type of key updated, it can be:"
 echo "           KSK | ZSK | H | R"
 echo "status - key status, old | new"
 echo "options - options to be applied to that key:"
 echo "          add | del | change"
 echo "Other variable options:"
 echo " \$DEBUG - Enable Debug!"
}

donsupdate()
{
if [ -e "${BINDDIR}/${ZONE}" ]; then
cd "${BINDDIR}/${ZONE}"
else
echo "Error zone dir doesn't exist!"
exit 1
fi

ZONEFILE="${ZONEPREFIX}${ZONE}"
ZSKLIST="/tmp/zsk_keys"

echo `ls -1 -c zsk/*.key` >& ${ZSKLIST}

case "${TYPE}" in
```

```
"ZSK")ZSKFILE="zsk/K${ZONE}"; FILE=${ZSKFILE};;
"H")HFILE="hradiusfile"; FILE=${HFILE};;
esac

case "${STATUS}" in
"old")
if [ "${TYPE}" = "ZSK" ]; then
ZSKTEMPFILE=`echo "${FILE}" | sed -e 's/\//\\\\\\\//g'`
EXTUSE=`cat ${ZSKLIST} | sed -e '2q' | tail -1 | sed -e "s/${ZSKTEMPFILE}\.//"`
else
EXTUSE=${OLDEXT}
fi
;;
"new")
if [ "${TYPE}" = "ZSK" ]; then
ZSKTEMPFILE=`echo "${FILE}" | sed -e 's/\//\\\\\\\//g'`
EXTUSE = `cat ${ZSKLIST} | sed -e '2q' | head -1 | sed -e "s/${ZSKTEMPFILE}\.//"`
else
EXTUSE=${NEWEXT}
fi
;;
*) echo "Error weird key status given"; exit 1;;
esac


case "${OPTS}" in
"add")
#Parse zone file
if [ -e ${ZONEFILE} ]; then
execute cat ${ZONEFILE} | grep "${FILE}.${EXTUSE}"
RES=$?;
if [ $RES -eq 1 ] && [ "${TYPE}" = "ZSK" ]; then
execute sed -i -e "/^.*zsk\/.*/i\\\$INCLUDE \"${FILE}.${EXTUSE}\"" ${ZONEFILE}
elif [ $RES -eq 1 ] && [ "${TYPE}" = "H" ]; then
execute sed -i -e "/^.*hradiusfile.*/i\\\$INCLUDE \"${FILE}.${EXTUSE}\"" ${ZONEFILE}
fi
else
echo "Zone file not found"
fi
;;
"del")
#Parse zone file
if [ -e ${ZONEFILE} ]; then
execute cat ${ZONEFILE} | grep "${FILE}.${EXTUSE}"
```

```
if [ $? -eq 0 ]; then
execute sed -i -e "s/^$INCLUDE.*\"${FILE}.${EXTUSE}\".*$//" ${ZONEFILE}
fi

execute unlink ${FILE}.${EXTUSE}
else
echo "Zone file not found"
exit 1
fi
;;
"change")
if [ "${TYPE}" = "H" ] || [ "${TYPE}" = "R" ]; then
if [ "${EXTUSE}" = "${OLDEXT}" ]; then
execute mv ${FILE}.${EXT} ${FILE}.${OLDEXT}
execute mv ${FILE}.${NEWEXT} ${FILE}.${EXT}
else
execute mv ${FILE}.${EXT} ${FILE}.${NEWEXT}
execute mv ${FILE}.${OLDEXT} ${FILE}.${EXT}
fi
fi
if [ "${TYPE}" = "ZSK" ] || [ "${TYPE}" = "KSK" ]; then
execute touch ${FILE}.${EXTUSE} #makes this new key
fi
;;

*) echo "Error weird option given"; exit 1;;
esac

execute unlink ${ZSKLIST}

cd ${PWD}
}

if [ ! $# -eq 4 ]; then
 echo "Please especify all necessary parameters"
 help
 exit 1
fi

if [ "$1" = "--help" ]; then
help
exit 0
fi

#Check Root
```

```
if [ ! "${USER}" = "root" ]; then
echo "You are not root user, using sudo!"
echo "Check if you are in sudoers file!"
SUDO_NEEDED=1
fi

#Zone name
ZONE=$1

until [ -z "$2" ]; do
        case "$2" in
          "KSK")echo "Not Implemented, future work"; exit 1;;
          "ZSK")
              TYPE="$2"
              STATUS="$3"
              OPTS="$4";;
          "H")
              TYPE="$2"
              STATUS="$3"
              OPTS="$4";;
          "R") echo "Not Implemented, future work"; exit 1;;
           "--help") help; exit 0;;
          *)  ;; #Ignore
        esac
        shift
done

if [ ! -z ${TYPE} ] && [ ! -z ${STATUS} ] && [ ! -z ${OPTS} ]; then
donsupdate
else
echo "Some parameters are missing!"
exit 1
fi
```

binddsupdater - Script de boot

```
#! /bin/sh
#
#  Generated: Fri Jun  6 04:47:32 MDT 2008
#
#  binddsupdater.sh - This little Script ensures, DNS RRs can be received
#
#####################################################################

set -e
```

```
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

if [ -r /lib/lsb/init-functions ]; then
. /lib/lsb/init-functions
logbegin="log_begin_msg"
logend="log_end_msg"
else
logbegin="echo -n"
logend=`printf "echo .\n"`
fi

# Exit if the daemon binary is NOT available, executable, etc.
test -x /usr/bin/binddsupdater || exit 0

# Start function
d_start() {
/usr/bin/binddsupdater create
}
# Stop function
d_stop() {
/usr/bin/binddsupdater delete
}

case "$1" in
start)
$logbegin "Starting binddsupdater..."
d_start
$logend $?
;;
stop)
$logbegin "Stopping binddsupdater..."
d_stop
$logend $?
;;
restart)
$0 stop
sleep 1
$0 start
;;
*)
log_success_msg "Usage: $0 {start|stop|restart}"
exit 1
;;
esac
exit 0
```

Anexos D

RADIUS

radiusupdate - Script de instalação da framework

```
#!/bin/bash

USER=`whoami`
: ${DEBUG:=0}
SUDO_NEEDED=0


CERTOVERLAY="/var/radius"
radiusupdate_monthly="${CERTOVERLAY}/radiusupdate-monthly"
certupdater="${CERTOVERLAY}/certupdater"
tasks_file="${CERTOVERLAY}/zones"

#Used for execute a command and prints the command used if DEBUG mode is set
execute() {
if [ ${DEBUG} -eq 1 ]; then
echo "$@"
fi
#Check if commands are executed in root or sudo mode
if [ ${SUDO_NEEDED} -eq 1 ]; then
sudo $@
else
$@
fi
}

help()
{
 echo "Usage:"
 echo "radiusupdate <dns zone> <options>"
 echo "Specifies to wich dns zone the radius server is depending on"
 echo "It criates rules and tasks for a first time use"
 echo "mirror - it sends a copy"
```

```
 echo "Other variable options:"
 echo " \$DEBUG - Enable Debug!"
}

first_usetasks()
{

#Check dir existance
if [ ! -d ${CERTOVERLAY} ]; then
execute mkdir ${CERTOVERLAY}
fi

if [ ! -e ${CERTOVERLAY}/radiusupdate ];then
execute cp radiusupdate ${CERTOVERLAY}
execute chmod +x ${CERTOVERLAY}/radiusupdate
fi
if [ ! -e ${CERTOVERLAY}/generate_certs ];then
execute cp generate_certs ${CERTOVERLAY}
execute chmod +x ${CERTOVERLAY}/generate_certs
fi
if [ ! -e ${CERTOVERLAY}/ca.cnf ] || [ ! -e ${CERTOVERLAY}/server.cnf ]; then
execute cp ca.cnf ${CERTOVERLAY}
execute cp server.cnf ${CERTOVERLAY}
fi

if [ ! -e ${CERTOVERLAY}/certupdater ];then
execute cp certupdater ${CERTOVERLAY}
execute chmod +x ${CERTOVERLAY}/certupdater
fi

if [ ! -e ${CERTOVERLAY}/dnskey ];then
execute cp dnskey ${CERTOVERLAY}
execute chmod +x ${CERTOVERLAY}/dnskey
fi

if [ ! -e ${CERTOVERLAY}/radiusupdate-monthly ]; then
execute cp radiusupdate-monthly ${CERTOVERLAY}
execute chmod +x ${CERTOVERLAY}/radiusupdate-monthly
fi

if [ ! -e ${CERTOVERLAY}/put_dnskey ];then
execute cp put_dnskey ${CERTOVERLAY}
execute chmod +x ${CERTOVERLAY}/put_dnskey

fi
```

```
if [ ! -e ${tasks_file} ]; then
execute touch ${tasks_file}
fi
}

if [ $# -lt 1 ]; then
 echo "Please especify to wich zone the tasks are met"
 help
 exit 1
fi

if [ "$1" = "--help" ]; then
help
exit 0
fi

#Check Root
if [ ! "${USER}" = "root" ]; then
echo "You are not root user, using sudo!"
echo "Check if you are in sudoers file!"
SUDO_NEEDED=1
fi

ZONE="$1"

first_usetasks

if [ ! -z $2 ] && [ "$2" = "mirror" ];then
#Check if zone file is in tasks_file
execute cat ${tasks_file} | grep "${ZONE}" > /dev/null 2>&1
RES=$?

if [ ${RES} -eq 1 ]; then
echo "$ZONE" | execute tee -a ${tasks_file} > /dev/null 2>&1
echo "$ZONE added to database"

else
echo "Zone is already in database!"
fi
elif [ ! -z $2 ] && [ ! "$2" = "mirror" ]; then
echo "Second option is unknown!"
exit 1
else
#Reset tasks file
```

```
execute rm ${tasks_file}
execute touch ${tasks_file}
#Add main zone to task zones
echo "$ZONE" | execute tee -a ${tasks_file} > /dev/null 2>&1
#Update Crontab tasks file
execute ${radiusupdate_monthly} 0 > /dev/null 2>&1
fi
```

radiusupdate_monthly - Script responsável pelas acções mensais

```
#!/bin/bash

DATE=`date +%x`
DATEDAY=`date +%D | awk -F'/' '{print $2}'`
DATEMON=`date +%D | awk -F'/' '{print $1}'`

TIMEMINUTC=`date -u +%R | awk -F':' '{print $2}'`
TIMEHOURUTC=`date -u +%R | awk -F':' '{print $1}'`

TIMEMIN=`date +%R | awk -F':' '{print $2}'`
TIMEHOUR=`date +%R | awk -F':' '{print $1}'`

HOURFIX=02
MINFIX=05

UPDATE_FILE="/tmp/.crontabupdate"
TTL=6 #(hours)

RADIUS_SERVICE="freeradius"

CERTOVERLAY="/var/radius"
certupdater="${CERTOVERLAY}/certupdater"
radiusupdate_monthly="${CERTOVERLAY}/radiusupdate-monthly"

debug()
{
echo "SET: | ${TIMEMIN} ${TIMEHOUR} | ${DATEDAY} ${DATEMON} |"
echo "USE: | ${MINTOUSE} ${HOURTOUSE} | ${DAYTOUSE} ${MONTHTOUSE} |"
}

validate_day()
{
#Specify how much days each month have
MON31=( "1" "3" "5" "7" "8" "10" "12" )
MON30=( "4" "6" "9" "11" "0" "0" )
```

```
MON28=( "2" "0" "0" "0" "0" "0" )
#Arguments
DAY=$1
MONTH="$2"

#optimization 28 days rule
if [ ${DAY} -le 28 ]; then
return ${DAY};
fi

for i in `seq 0 6`;
do
if [ "${MON31[$i]}" = "${MONTH}" ]; then
return ${DAY};
fi

if [ "${MON30[$i]}" = "${MONTH}" ]; then
if [${DAY} -le 30 ]; then
return ${DAY};
else #Normalized day to 30
return 30;
fi
fi

if [ "${MON28[$i]}" = "${MONTH}" ]; then
if [${DAY} -gt 28 ]; then
#Normalized day to 28
return 28;
fi
fi
done

echo "Don't know what happened here!"
return 1;
}

getdays_month()
{
#Specify how much days each month have
MON31=( "1" "3" "5" "7" "8" "10" "12" )
MON30=( "4" "6" "9" "11" "0" "0" )
MON28=( "2" "0" "0" "0" "0" "0" )
#Arguments
MONTH="$1"
```

```
for i in `seq 0 6`;
do
if [ "${MON31[$i]}" = "${MONTH}" ]; then
return 31;
fi

if [ "${MON30[$i]}" = "${MONTH}" ]; then
return 30;
fi

if [ "${MON28[$i]}" = "${MONTH}" ]; then
return 28;
fi
done

echo "Don't know what happened here!"
return 28;
}

validate_adding()
{

if [ $HOURTEMP -ge 24 ]; then
let "DAYTEMP = DATEDAY + 1" # 1 day
let "HOURTOUSE = HOURTEMP - 24" # 24h time

#Check if day is valid for that month, because of february 28 days
validate_day ${DAYTEMP} ${DATEMON}
DAYTOUSE=$?

if [ ! "${DAYTEMP}" = "${DAYTOUSE}" ]; then
DAYTOUSE=1; #begining of month
#Increase a month
let "MONTHTOUSE = DATEMON - 1" # 1 month
if [ ${MONTHTOUSE} -gt 12 ]; then
#Revert a year in months
let "MONTHTOUSE -= 12"
fi
fi
else
HOURTOUSE=${HOURTEMP}
fi
}

validate_subtracting()
```

```
{

if [ $HOURTEMP -lt 0 ]; then
let "DAYTOUSE = DATEDAY - 1" # 1 day
let "HOURTOUSE = HOURTEMP + 24" # 24h time
if [ ${DAYTOUSE} -le 0 ]; then
#Reverte a month
let "MONTHTOUSE = DATEMON - 1" # 1 month
if [ ${MONTHTOUSE} -le 0 ]; then
#Revert a year
MONTHTOUSE=12;
fi
#Get last day of that month
getdays_month ${MONTHTOUSE}
DAYTOUSE=$?
fi
else
HOURTOUSE=${HOURTEMP}
fi
}

sync_time()
{
#Set default values
DAYTOUSE=${DATEDAY}
MONTHTOUSE=${DATEMON}
MINTOUSE=${TIMEMIN}
HOURTOUSE=${TIMEHOUR}

#Tasks to be executed at a fixed hours
DIFFMIN=0
DIFFHOUR=0
CARRY=0
#Calculate time diferences
let "DIFFMIN = TIMEMIN - TIMEMINUTC"
let "DIFFHOUR = TIMEHOUR - TIMEHOURUTC"

#Fixable update hour to 02h10
TIMEMIN=${MINFIX}
let "MINTOUSE = TIMEMIN + DIFFMIN"
if [ $MINTOUSE -lt 0 ]; then
let "MINTOUSE += 60"
CARRY="-1"
elif [ $MINTOUSE -gt 59 ]; then
let "MINTOUSE -= 60"
```

```
CARRY="1"
fi
TIMEHOUR=${HOURFIX};
let "HOURTEMP = TIMEHOUR + DIFFHOUR + CARRY"
if [ $DIFFHOUR -lt 0 ]; then
echo "local hour is less $DIFFHOUR hours than UTC time"
validate_subtracting
elif [ $DIFFHOUR -eq 0 ]; then
echo "local hour is the same as UTC time"
else
echo "local hour is more $DIFFHOUR hours than UTC time"
validate_adding
fi

#Use new dates
DATEDAY=${DAYTOUSE}
DATEMON=${MONTHTOUSE}
TIMEMIN=${MINTOUSE}
TIMEHOUR=${HOURTOUSE}
}

add_ttl()
{

#Add 1 TTL from Day Time
let "HOURTEMP = TIMEHOUR + TTL" # 24h + 1 TTL each time

validate_adding

}

addmonth()
{
#Actualiza o mês para o proximo mes
let "MONTHTOUSE = DATEMON + 1" # 12 months plus one

#Normalized month
if [ ${MONTHTOUSE} -gt 12 ]; then
let "MONTHTOUSE -= 12" # 12 months
fi

#Check if day is valid for that month, because of february 28 days
validate_day ${DATEDAY} ${MONTHTOUSE}
DAYTOUSE=$?
```

```
}

remove_2ttl()
{
#Remove 2 TTL from Day Time
let "HOURTEMP = TIMEHOUR - (TTL + TTL)" # 24h - 2 TTL each time

validate_subtracting $HOURTEMP
}

monthly_tasks()
{
case "${TASK}" in
#Add Crontab rule
"0")
#It's first time interation, next interation will be in 2 TTLS
addmonth
remove_2ttl

#Generate Certs
${certupdater} old create all #> /dev/null 2>&1
${certupdater} old set default #> /dev/null 2>&1
${certupdater} old set dnskey #> /dev/null 2>&1
#Update Crontab tasks file
echo "SHELL=/bin/bash" >& ${UPDATE_FILE}
echo "PATH=${CERTOVERLAY}:${PATH}" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "${TIMEMIN} ${TIMEHOUR} * * * ${certupdater} old create server > /dev/null 2>&1
echo "${TIMEMIN} ${HOURTOUSE} ${DAYTOUSE} ${MONTHTOUSE} * ${radiusupdate_monthly} 1
crontab ${UPDATE_FILE} > /dev/null 2>&1
unlink ${UPDATE_FILE}
#Reload RADIUS Server
invoke-rc.d ${RADIUS_SERVICE} restart;;

#Executado 2 TTL's antes
"1")
add_ttl
add_ttl

#Recriate RADIUS CA Certificate and consequentally it's RADIUS server certificate
${certupdater} new create all #> /dev/null 2>&1
${certupdater} new set dnskey #> /dev/null 2>&1
#edit Crontab with new file
echo "SHELL=/bin/bash" >& ${UPDATE_FILE}
```

```
echo "PATH=${CERTOVERLAY}:${PATH}" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "${TIMEMIN} ${HOURTOUSE} ${DAYTOUSE} ${MONTHTOUSE} * ${radiusupdate_monthly} 2 > /de
crontab ${UPDATE_FILE} > /dev/null 2>&1
unlink ${UPDATE_FILE};;

#Executado on scheldule
"2")
#Set next time interation, will be one month less 2 TTLS
addmonth
remove_2ttl

#updater
${certupdater} old del #> /dev/null 2>&1
${certupdater} new set old #> /dev/null 2>&1
${certupdater} old set default #> /dev/null 2>&1

#Update Crontab tasks file
echo "SHELL=/bin/bash" >& ${UPDATE_FILE}
echo "PATH=${CERTOVERLAY}:${PATH}" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "" >> ${UPDATE_FILE}
echo "${TIMEMIN} ${TIMEHOUR} * * * ${certupdater} old create server > /dev/null 2>&1" >>
echo "${TIMEMIN} ${HOURTOUSE} ${DAYTOUSE} ${MONTHTOUSE} * ${radiusupdate_monthly} 1 > /de
crontab ${UPDATE_FILE} > /dev/null 2>&1
unlink ${UPDATE_FILE}
#Reload RADIUS Server
#/etc/init.d/${RADIUS_SERVICE} restart #> /dev/null 2>&1
invoke-rc.d ${RADIUS_SERVICE} restart;;
*) echo "Unknown fase!"; exit 1;;
esac
}

if [ ! $# -eq 1 ]; then
 echo "Please especify what certs and task should be performed"
 exit 1
fi

TASK=$1;

sync_time

monthly_tasks
```

put_dnskey - Script que coloca remotamente os registos H

```
#!/bin/bash

RUSER="pmaia"
RHOST="${RUSER}@172.16.10.6" # m6
RBINDDIR="/var/tmp/radiusDS"
HFILE="hradiusfile.ds"
BINDDIR="/etc/bind/"
CERTDIR="/etc/freeradius/certs"
CACERTDIR=$CERTDIR"/CA"

USER=`whoami`
PWD=`pwd`
: ${DEBUG:=0}
SUDO_NEEDED=0

FIXEDIP="8"
FIXEDFQDN="radius"

#Used for execute a command and prints the command used if DEBUG mode is set
execute() {
if [ ${DEBUG} -eq 1 ]; then
echo "$@"
fi
#Check if commands are executed in root or sudo mode
if [ ${SUDO_NEEDED} -eq 1 ]; then
sudo "$@"
else
"$@"
fi

}

help()
{
 echo "Usage:"
 echo "put_dnskey <zone name> -remote"
 echo "remote - Places the key file in a remote host"
 echo "Other variable options:"
 echo " \$DEBUG - Enable Debug!"
 echo ""
 echo "Please note that you need to have a 'regular' directory map"
}
```

```
if [ $# -eq 0 ]; then
 echo "Please especify what certs want to generate"
 help
 exit 1
fi

if [ "$1" = "--help" ]; then
help
exit 0
fi

#Check Root
if [ ! "${USER}" = "root" ]; then
echo "You are not root user, using sudo!"
echo "Check if you are in sudoers file!"
SUDO_NEEDED=1
fi

#Root protected dir
execute test -d ${CACERTDIR}
if [ $? -eq 1 ]; then
echo "CA certificate dir, doesn't exist!"
help
exit 1
fi

if [ ! -z ${2} ] && [ ! "$2" = "-remote" ]; then
echo "Unkown second option"
help
exit 1
fi

if [ ! -d "${BINDDIR}/${ZONE}" ] && [ ! "$2" = "-remote" ]; then
echo "Domain zone dir, doesn't exist!"
help
exit 1
fi

# add our domain
ZONE=$1;

echo "Add DNSKEY field to our zone"

#Get CA public Key hash
execute getCAPKey ${CACERTDIR}/cacert.pem tmpdns.ds > /dev/null 2>& 1
```

```
#Clean Delimiters!?
echo "${ZONE}" | grep ".in-addr.arpa" >& /dev/null 2>1
REVERSE=$?
if [ ${REVERSE} -eq 0  ]; then
execute sed -i -e "1i\\${FIXEDIP}.${ZONE}.\t" tmpdns.ds
else
execute sed -i -e "1i\\${FIXEDFQDN}.${ZONE}.\t" tmpdns.ds
fi
execute tr -d '\n' < tmpdns.ds >& ${HFILE}

execute sed -i -e '$a\' ${HFILE}

if [ -e ${HFILE} ]; then
if [ -z "$2" ] && [ "$2" = "-remote"]; then
cp ${HFILE} ${BINDDIR}/${ZONE}
else
ssh ${RHOST} "mkdir ${RBINDDIR}/${ZONE} >& /dev/null 2>1"
scp ${HFILE} ${RHOST}:${RBINDDIR}/${ZONE}/${HFILE}

fi
fi

echo "Cleaning files"
execute rm tmpdns.ds
execute rm ${HFILE}
```

certupdater - Script que trata do rollover dos certificados

```
#!/bin/bash

RADIUSDIR="/etc/freeradius/"
RADIUSCERTDIR="${RADIUSDIR}/certs"

CERTOVERLAY="/var/radius"
OLDCERTDIR="oldcerts"
NEWCERTDIR="newcerts"
CERTDIR="${CERTOVERLAY}/certs"
tasks_file="${CERTOVERLAY}/zones"

USER=`whoami`
: ${DEBUG:=0}
SUDO_NEEDED=0
PWD=`pwd`
```

```
#Used for execute a command and prints the command used if DEBUG mode is set
execute() {
if [ ${DEBUG} -eq 1 ]; then
echo "$@"
fi
#Check if commands are executed in root or sudo mode
if [ ${SUDO_NEEDED} -eq 1 ]; then
sudo "$@"
else
"$@"
fi
}

help()
{
 echo "Usage:"
 echo "certupdater <cert overlay> <options> <status>"
 echo "Cert overlay - if it's the old | new overlay"
 echo "options - options to be applied to that key:"
 echo "          create | del | set"
 echo "status - status options for each option given:"
 echo "          default | old | new | dnskey | all | server"
 echo "Other variable options:"
 echo " \$DEBUG - Enable Debug!"
}


if [ ! $# -eq 3 ]; then
 echo "Please especify all necessary parameters"
 help
 exit 1
fi

if [ "$1" = "--help" ]; then
help
exit 0
fi


#Check Root
if [ ! "${USER}" = "root" ]; then
echo "You are not root user, using sudo!"
echo "Check if you are in sudoers file!"
SUDO_NEEDED=1
fi
```

```
OVERLAY=$1
OPTS=$2
STATUS=$3

case "${OVERLAY}" in
"old")TEMPDIR=${CERTOVERLAY}/${OLDCERTDIR};;
"new")TEMPDIR=${CERTOVERLAY}/${NEWCERTDIR};;
*) echo "Error weird overlay order given"; exit 1;;
esac

case "${OPTS}" in
"create")
if [ ! -e ${TEMPDIR} ]; then
execute mkdir ${TEMPDIR}
fi

cd ${TEMPDIR}

if [ ! -e generate_certs ] || [ ! -e ca.cnf ] || [ ! -e server.cnf ]; then
execute cp ${CERTOVERLAY}/generate_certs ${TEMPDIR} #> /dev/null 2>1
execute cp ${CERTOVERLAY}/ca.cnf ${TEMPDIR} #> /dev/null 2>1
execute cp ${CERTOVERLAY}/server.cnf ${TEMPDIR} #> /dev/null 2>1
fi
;;
"del")
if [ -e ${TEMPDIR} ]; then
execute rm -r ${TEMPDIR}
fi
exit 0 ;;
"set")
if [ ! -e ${TEMPDIR} ]; then
execute mkdir ${TEMPDIR}
fi

cd ${CERTOVERLAY}

DEST=${RADIUSCERTDIR}
;;
*) echo "Error weird options order given"; exit 1;;
esac

case "${STATUS}" in
"default")
```

```
if [ -e ${CERTDIR} ];then
execute rm ${CERTDIR}
fi

execute ln -sf ${TEMPDIR} ${CERTDIR}

if [ ! "${DEST}" = "" ]; then
if [ -e ${DEST}/servcert.pem ]; then
execute rm ${DEST}/servcert.key #> /dev/null 2>1
fi
execute ln -sf ${CERTDIR}/servcert.pem ${DEST}
execute chown root ${DEST}/servcert.pem
execute chgrp freerad ${DEST}/servcert.pem
if [ -e ${DEST}/servcert.key ]; then
execute rm ${DEST}/servcert.key #> /dev/null 2>1
fi
execute ln -sf ${CERTDIR}/servcert.key ${DEST}
execute chown root ${DEST}/servcert.key
execute chgrp freerad ${DEST}/servcert.key
if [ -e ${DEST}/CA ]; then
execute rm -r ${DEST}/CA #> /dev/null 2>1
fi
execute ln -sf ${CERTDIR}/CA ${DEST}
execute chown root ${DEST}/CA
execute chgrp freerad ${DEST}/CA
else
echo "set command not given";
exit 1;
fi ;;
"dnskey")
if [ "${OPTS}" = "set" ]; then
  execute ./dnskey ${tasks_file} > /dev/null 2>&1
else
echo "set command not given";
fi
;;
"old")
if [ -e ${TEMPDIR} ] && [ "${OVERLAY}" = "new" ]; then
execute mv ${TEMPDIR} ${CERTOVERLAY}/${OLDCERTDIR}
fi
;;
"new")
if [ -e ${TEMPDIR} ] && [ "${OVERLAY}" = "old" ]; then
execute mv ${TEMPDIR} ${CERTOVERLAY}/${NEWCERTDIR}
fi
```

```
;;
"all") execute ./generate_certs ${STATUS} ;;
"server") execute ./generate_certs ${STATUS} ;;
*) echo "Error weird status order given"; exit 1;;
esac

cd ${PWD}
```