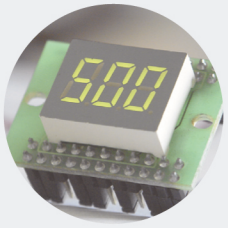
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Migração de ligações Wi-Fi não seguras para ligações seguras, após autenticação em captive portal

ANDRÉ ALEXANDRE GOMES FERNANDES
Novembro de 2015

POLITÉCNICO
DO PORTO

MIGRAÇÃO DE LIGAÇÕES WI-FI NÃO SEGURAS PARA LIGAÇÕES SEGURAS, APÓS AUTENTICAÇÃO EM CAPTIVE PORTAL

André Alexandre Gomes Fernandes

Departamento de Engenharia Electrotécnica

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização em Telecomunicações

2015

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: André Alexandre Gomes Fernandes, Nº 1100387, 1100387@isep.ipp.pt
Orientação científica: Professor Doutor Jorge Botelho da Costa Mamede, jbm@isep.ipp.pt
Empresa: INESC TEC - Porto
Supervisão: Professor Doutor António Alberto dos Santos Pinto, apinto@inescporto.pt

Departamento de Engenharia Electrotécnica

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização em Telecomunicações

2015

“A segurança de um criptosistema
deve estar na chave e não no secretismo de uma parte do sistema”
Auguste Kerckhoffs

Agradecimentos

O meu sincero agradecimento a todos os que diretamente ou indiretamente me ajudaram na elaboração desta Tese contribuindo para o sucesso da mesma.

Aos meus pais agradeço o apoio incondicional e a sua presença. Os valores que me transmitiram serviram de base para me tornar o que sou, sem eles não seria possível.

Agradeço também ao Professor Doutor Jorge Mamede, por me ter introduzido o tema tratado, pelo apoio contínuo e orientação, não só na elaboração da Tese, como desde o início do meu percurso universitário.

Ao Professor Doutor António Pinto, por todas a palavras de incentivo, conselhos, acompanhamento e paciência demonstrados no desenvolvimento da Tese.

Aos meus amigos pelo companheirismo e amizade nas diversas fases da vida académica.

À entidade envolvida e aos seus colaboradores, INESC TEC.

Um obrigado especial, à minha namorada, Rita Monteiro, que esteve sempre presente nos momentos mais complicados e me incentivou na elaboração da Tese.

A todos vocês, o meu muito obrigado.

André Fernandes.

Resumo

Atualmente a popularidade das comunicações Wi-Fi tem crescido, os utilizadores acedem a partir de vários dispositivos como telemóveis, *tablets*, computadores portáteis sendo estes utilizados por qualquer pessoa nos mais variados locais. Com esta utilização massiva por parte dos utilizadores surgiram os *hotspots* Wi-Fi públicos (em aeroportos, estações de comboios, etc) que permitem a ligação de clientes recorrendo a ligações wireless não seguras (ou abertas). Tais *hotspots* utilizam, após a ligação de um cliente, um *captive portal* que captura o tráfego IP com origem no cliente e o redireciona para uma página *Web* de entrada. A página *Web* permite ao cliente comprar tempo de acesso à Internet ou, caso já seja um cliente da empresa, autenticar-se para ter acesso à Internet.

A necessidade da ligação aberta assenta na possibilidade do operador do *hotspot* vender acesso à Internet a utilizadores não conhecidos (caso contrário teria de fornecer-lhes uma senha previamente). No entanto, fornecer um acesso à Internet *wireless* sem qualquer tipo de segurança ao nível físico permite que qualquer outro utilizador consiga obter informação sobre a navegação *Web* dos utilizadores ligados (ex.: escuta de pedidos DNS).

Nesta tese pretende-se apresentar uma solução que estenda um dos atuais mecanismos de autenticação Wi-Fi (WPA, WPA2) para que permita, após autenticação em *captive portal*, a migração de uma ligação aberta para uma ligação segura.

Palavras-Chave

Comunicações Wi-Fi, autenticação, captive portal, ligação segura.

Abstract

Nowadays, the popularity of Wi-Fi communications has grown because the users can access networks from multiple devices such as mobile phones, tablets, laptops and these being used by anyone in different places all over the world. With this massive use of technologies, Public Wi-Fi hotspots such as airports, train stations, etc. created allowing the connection through unsecured (or open) connections. After the connection with the client, these hotspots use a captive portal that captures the client IP traffic and redirects it to an particular Web page. The *website* allows customers to buy Internet access time or, if they already have credentials from that company, log-in and access the Internet.

The need for open connection is based on the possibility of the hotspot operator to sell Internet accesses to unknown users (otherwise would have to provide them with a password previously). However, to provide a wireless access to the Internet without any kind of security at the physical level allows any user can get information about web browsing of connected users (eg.: listening DNS requests).

This thesis is intend to provide a solution based on the extension of the current mechanisms for Wi-Fi authentication (WPA, WPA2) to allow, after an authentication in captive portal, to migrate from an open connection to a secure connection.

Keywords

Wi-Fi communications, authentication, captive portal, secure connection.

Índice

Índice de Figuras

Índice de Tabelas

Índice de Código

Acrónimos

AAA – Authentication, Authorization and Accounting

AES – Advanced Encryption Standard

AP – Access Point

CHAP – Challenge Handshake Protocol

DES – Data Encryption Standard

DHCP – Dynamic Host Configuration Protocol

DNS – Domain Name System

EAP – Extensible Authentication Protocol

HTTP – Hypertext Transfer Protocol

IEEE – Institute of Electrical and Electronics Engineers

IDEA – International Data Encryption Algorithm

IP – Internet Protocol

JavaSE – Java Standard Edition

LAN – Local Area Network

MAC – Medium Acess Control

NAS – Network Acess Server

NAT – Network Address Translation

NIST – National Institute of Standards and Technology

OSI – Open Systems Interconnection

PAP – Password Authentication Protocol

PHP – Hypertext Preprocessor

PPTP – Point-to-point tunneling protocol

RADIUS – Remote Authentication Dial In User Service

RFC – Request For Comments

RSA – Rivest Shamir Adleman

SHA – Secure Hash Algorithm

SSL – Secure Sockets Layer

TCP – Transmission Control Protocol

TTLS – Tunneled Transport Layer Security

UAM – Universal Access Method

WEP – Wired Equivalent Privacy

WLAN – Wireless Local Area Network

WPA – Wi-Fi Protected Access

WWW – World Wide Web

1. INTRODUÇÃO

Hoje em dia nos aeroportos ou centro-comerciais existe a possibilidade de aceder à Internet através de pontos de acesso de redes sem fios. No entanto, quando se acede a estes *hotspots* Wi-Fi públicos a ligação que se obtém não é segura estando por isso aberta, isto é, exposta a terceiros. Nestes *hotspots*, depois de um utilizador tentar aceder a um determinado *website* é redirecionado para uma página onde lhe pode ser exigido os dados de autenticação, pagamento, leitura e aceitação das normas de utilização, etc.

Pelo facto desta ligação ser aberta, é sabido que, fornecer um acesso à rede wireless sem qualquer tipo de segurança ao nível físico, compromete a segurança da rede, possibilitando que qualquer utilizador obtenha informação relativamente à navegação de outro utilizador que também esteja ligado a essa rede. (ex: escuta dos pedidos de *Domain Name System (*DNS)). Com a massiva utilização da Internet por parte dos utilizadores, e a preocupação natural dos utilizadores pela sua segurança, é normal que os utilizadores de *hotspots* públicos pretendam sentir-se seguros enquanto navegam na *Internet*. Pretende-se então implementar uma solução para este problema.

A Figura 1, representa o cenário de referência. A sua representação contém três entidades distintas: o utilizador, o operador *captive portal* e o operador de autenticação. O utilizador é a entidade que pretende ter acesso à Internet e que, normalmente, está na posse de um conjunto de credenciais válidas. Essas credenciais são compartilhadas entre o utilizador e o operador de autenticação. O operador *captive portal* é a entidade que implementa a rede local e permite a ligação do utilizador. O operador de autenticação é a

entidade que possui a base de dados com as credenciais dos utilizadores, sendo que, é suposto existir previamente um acordo estabelecido com o *captive portal*.

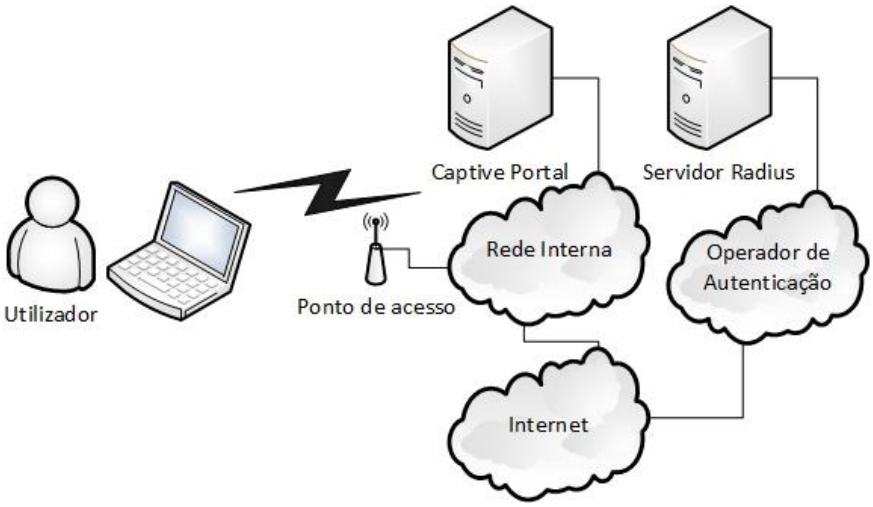


Figura 1 – Representação do cenário de referência.

Apesar de permitir que os utilizadores tenham facilidade na ligação e na utilização, a adoção de ligações sem fio abertas apresenta alguns inconvenientes no que se refere à segurança, privacidade e confidencialidade. Qualquer utilizador na proximidade de um *hotspot* pode capturar todo o tráfego referente a todos os utilizadores do *hotspot*. Este facto pode levar (i) a perda de confidencialidade no caso dos protocolos de aplicação serem utilizados através de ligações inseguras (email ou navegação na *Web* por exemplo); a (ii) perda de segurança, se as credenciais do utilizador são trocadas em "*clear-text*" enquanto este navega pela *Web*; e (iii) perda de privacidade, se a rede utiliza um serviço de DNS padrão em todas as consultas DNS em "*clear-text*".

Uma solução possível para minimizar os problemas referidos seria a utilização de formas mais seguras de ligações sem fio, como *Wi-Fi Protected Access* (WPA).[1] No entanto, o processo de configuração WPA constitui um problema já que é visto como uma forma de impedir a utilização do *captive portal*. A solução proposta aborda esta questão, permitindo tornar transparente e automática a configuração de ligações sem fios seguras, para utilizares com a autenticação no *captive portal* bem-sucedida.

1.1. MOTIVAÇÃO

Os utilizadores a quando da sua utilização do *captive portal*, para além de comodidade, facilidade e rapidez de acesso pretendem que a sua utilização seja segura. Assim, pretende-se implementar uma solução para estender os atuais mecanismos de autenticação *wireless* (WPA e WPA2) para que seja possível após autenticação em *captive portal*, a migração de uma ligação aberta para uma ligação segura.

O que se pretende, é que após a autenticação ser realizada com sucesso, a ligação seja, a partir desse instante, segura e por esse motivo, o utilizador não se sinta reticente na sua navegação.

1.2. OBJETIVO

O objetivo desta tese consiste no estudo, implementação e avaliação da segurança Wi-Fi após autenticação em *captive-portal*, nomeadamente:

- Especificação de uma extensão a um protocolo de autenticação Wi-Fi para permitir uma migração de segurança para ligações Wi-Fi abertas.

- Teste e validação de um protótipo laboratorial.

O objetivo principal deste projeto é:

- Permitir que o utilizador possa usufruir de uma sessão segura após a sua autenticação em pontos de acesso públicos.

1.3. ORGANIZAÇÃO DO RELATÓRIO

Este documento é estruturado da seguinte forma: capítulo 2 apresenta o trabalho relacionado, protocolos de rede relevantes e descreve a operação comum de um *hotspot* com *captive portal*; capítulo 3 é relativo a criptografia aplicada; capítulo 4 detalha a solução proposta, descreve a arquitetura e implementação da prova de conceito do protótipo final; o capítulo 5 apresenta os resultados obtidos; o capítulo 6 conclui este documento.

2. SEGURANÇA EM REDES IP SEM FIOS

Wireless Local Area Networks (WLAN)[2] suporta tanto o estabelecimento de uma ligação sem fios não segura como uma segura. A ligação não segura refere-se habitualmente a uma ligação sem fio aberta. Os utilizadores na proximidade, ou mais longe, no caso da utilização de antenas direcionais altamente sensíveis, podem ligar-se e escutar todo o tráfego transmitido.

A ligação sem fios pode ser protegida, adicionando funcionalidades como autenticação de cliente e criptografia de mensagens. *Wired Equivalent Privacy* (WEP) é uma autenticação de chave compartilhada baseada num mecanismo de desafio-resposta unilateral que também suporta criptografia da ligação sem fios. WEP foi projetado para fornecer a segurança equivalente a uma LAN com fio por meio da criptografia de tráfego de rede com o algoritmo RC4 [3]. Posteriormente, o *draft* IEEE 802.11i, que define o *Wi-Fi Protected Access* (WPA), foi definido para resolver vários problemas de segurança identificados com WEP.

O WPA foi introduzido pela *Wi-Fi Alliance* (WFA) como a substituição do WEP. O desenvolvimento de WPA foi orientado no sentido de corrigir as falhas de segurança

encontradas no protocolo WEP. Entre as melhorias propostas, a mais significativa foi o uso do algoritmo RC4 de uma forma mais segura, dentro do protocolo *Temporal Key Integrity* Protocol (TKIP). Mais tarde, a WFA lançou a segunda versão do sucessor do WPA (WPA2), após a descoberta de algumas falhas de segurança no TKIP. Este foi substituído pelo protocolo *Counter Mode with Cipher Block Chaining Message Authentication Code Protocol* (CCMP) que utiliza o algoritmo de cifragem *Advanced Encryption Standard* (AES) simétrico.

O WPA, em qualquer versão, suporta dois modos distintos de funcionamento: WPA-Personal e WPA-Enterprise. WPA-Personal é análogo ao WEP, pois requer uma chave para o estabelecimento da ligação. Esta chave é uma chave que é compartilhada entre todos os utilizadores da rede sem fio e é conhecida como uma chave pré-compartilhada (*Pre-Shared Key* (PSK)). Por outro lado, WPA-Enterprise requer um conjunto de credenciais do utilizador para o estabelecimento da ligação e é implementado em conjunto com um servidor de autenticação, tal como um servidor RADIUS. WPA-Enterprise é habitual em ambientes de negócio em que existe um servidor RADIUS.

Extensible Authentication Protocol (EAP) é uma plataforma de autenticação. Esta fornece uma estrutura que permite a utilização segura de vários métodos de autenticação. *EAP Tunneled Transport Layer Security* (EAP-TTLS) é uma extensão do *EAP Transport Layer Security* (EAP-TLS), que foi criado para reduzir a complexidade de implementação TLS devido ao seu uso de certificados digitais. No TTLS, o servidor de autenticação só se autentica para o suplicante, e os clientes podem escolher autenticar-se no servidor. Portanto, é um método de autenticação que apresenta duas formas. EAP-TTLS suporta vários protocolos de autenticação de clientes internos como *Password Authentication Protocol* (PAP), *Challenge-Handshake Authentication Protocol* (CHAP), *Microsoft Challenge Handshake Authentication Protocol* (MSCHAP) e MSCHAPv2. O processo de autenticação ocorre dentro de um túnel seguro[4].

2.1.1. IEEE 802.1x

O IEEE 802.1x é uma norma do IEEE para o controlo de acessos na rede tendo em conta as portas. Faz parte do grupo de protocolos IEEE 802.1 e possibilita a autenticação de dispositivos associados a uma porta LAN, estabelecendo um ligação ponto a ponto ou

em caso falha de autenticação, negando o acesso a essa porta. É baseado no *Extensible Authentication Protocol* (EAP) [5].

É frequente o 802.1x ser implementado em pontos de acesso sem fios para superar as falhas de segurança do *Wired Equivalent Privacy* (WEP)[6] quando o ponto de acesso opera na rede privada. Geralmente, a autenticação é feita por uma terceira entidade, por exemplo o RADIUS, o que possibilita que apenas o cliente se autentique existindo autenticação de ambas as partes recorrendo a protocolos como EAP-TLS [7].

2.1.2. WI-FI PROTECTED ACCESS (WPA)

O WPA surge com o intuito de ultrapassar as vulnerabilidades do WEP. Com a alteração do WEP pelo WPA é possível melhorar a encriptação dos dados ao utilizar um protocolo de chave temporária (TKIP), que permite a criação de chaves por pacotes e contém ainda um mecanismo de deteção de erros, um vetor de inicialização de 48 bits em vez dos 24 bits que o WEP utiliza e um mecanismo de distribuição de chaves.

Uma das grandes vantagens é relativa à autenticação, que utiliza protocolo 802.1x e o EAP, que por intermédio de um servidor central faz autenticação de cada cliente ainda antes deste ter acesso à rede.

Uma das diferenças relativas ao WEP é a utilização de uma chave de encriptação dinâmica em detrimento da utilização da mesma chave de forma repetida. Esta característica do WPA não obriga a que as chaves de encriptação sejam introduzidas manualmente como acontece com o WEP [7].

2.1.3. WI-FI PROTECTED ACCESS 2 (WPA)

A segunda versão do WPA2 surgiu um ano mais tarde para ultrapassar as vulnerabilidades criptográficas do WPA quanto ao mecanismo TKIP. O WPA2 apresentou um algoritmo de cifra mais desenvolvido cujo nome é *Advanced Encryption Standard* (AES)[8], e o *Counter Mode with Cipher Block Chaining Message Authentication Code Protocol* (CCMP)[9]. Mantendo-se como inicialmente em WPA o modo pessoal e empresarial. O WPA2 apresentou novidades em relação ao WPA no que se refere à mobilidade, sendo que, incluí mecanismos para fazer reautenticação rápida de terminais e a possibilidade de fazer pré-autenticação [7]. Ao utilizar pré-autenticação esta informação é transmitida a partir do ponto de acesso que o cliente está utilizando para o ponto de acesso

de destino. A utilização desta configuração ajuda no processo de autenticação de clientes que se ligam a vários pontos de acesso em *roaming*.

2.1.4. EXTENSIBLE AUTHENTICATION PROTOCOL (EAP)

Permite a autenticação em redes sem fios e ligações ponto a ponto (PPP). Não se encontra restringido às redes sem fios pelo que pode ser utilizado em redes com fios, apesar deste tipo de utilização não ser tão frequente [10].

O EAP define o formato de mensagens autenticação, não sendo considerado um protocolo de rede. Cada protocolo que utiliza EAP seleciona o modo de encapsulamento das mensagens EAP dentro das mensagens do protocolo.

A sua importância é notória nas redes sem fios pois as credenciais do utilizador podem ser validadas pelo servidor RADIUS, caracterizando o processo de autenticação EAP na rede com mais controlo e a possibilidade de gestão dos acessos à rede. São vários os métodos de integração do EAP nas redes sem fios.

2.1.4.1. EAP-TRANSPORT LAYER SECURITY (EAP-TLS)

Relativamente ao EAP-TLS, um dos métodos EAP, a segurança do protocolo Transport Layer Security (TLS) é considerável desde que o utilizador perceba os avisos relativos a credenciais falsas. Este utiliza a *Public Key Infrastructure* (PKI) para tornar a ligação ao servidor de autenticação segura (ex: RADIUS). Uma das desvantagens conhecidas é o *overhead* concebido do lado do utilizador pelos certificados.

Ainda que não seja muito utilizado, é considerado uma das normas EAP mais seguras, e tem suporte universal de todos os fabricantes de *hardware* de redes sem fios.

2.1.4.2. EAP-TUNNELED TRANSPORT LAYER SECURITY (EAP-TTLS)

Este protocolo estende o TLS abordado em 2.1.4.1, é bastante utilizado em vários sistemas operativos apesar de não haver suporte nativo no sistema *Windows* para este protocolo EAP. Sendo para isso necessário a instalação de programas, tais como, o SecureW2, bastante conhecido.

EAP-TTLS [11] fornece segurança, o utilizador não precisa de se autenticar por um certificado para o servidor, o que facilita o processo de configuração visto que o cliente não precisa de o instalar.

Depois da validação por intermédio do certificado CA (*Certification Authority*), o servidor estabelece uma ligação segura (túnel) para a autenticação do cliente.

É possível utilizar-se um protocolo e estrutura existente alocando mecanismos de palavra-chave e base de dados de autenticação e o túnel seguro confere proteção aos ataques de escuta e interceção dos dados. A identidade do utilizador não é transmitida em aberto, sendo uma vantagem relativamente à privacidade [10].

2.1.4.3. PEAP

Foi proposto pela RSA security, Cisco System e Microsoft para uma norma aberta. Quanto à sua estrutura é idêntica à do EAP-TTLS, sendo necessário um certificado PKI no lado do servidor, para que seja criado um túnel TLS seguro para proteger a autenticação.

Este protocolo é referido como EAP-MSCHAPv2 [12]. Para o utilizador o processo de autenticação é bastante simples uma vez que é somente necessária a introdução do nome e senha para o servidor RADIUS proceder à validação [13].

2.1.5. CONCLUSÃO

A segurança de redes sem fios tem por objetivo a autenticação segura de utilizadores e proteção das comunicações. Quanto à autenticação, a norma mais relevante é o IEEE 802.1x, sendo a base o EAP. Estas normas são *frameworks* de autenticação para redes sem fios. O protocolo IEEE 802.1x foi numa fase inicial desenvolvido para redes com cabos, pelo que a sua adaptação a redes sem fios criou alguns problemas devido à maior facilidade em intercetar e escutar as comunicações. Isto levou a que tenham surgido novos métodos EAP baseados em túneis TLS. Primeiro surgiu o EAP-TLS, método que soluciona a grande maioria dos problemas do EAP em redes sem fios, e por isso, é considerado por muitos como o método mais seguro. Apesar disso, obriga a que tanto o cliente EAP como o servidor tenham certificados instalados, o que prefigura uma desvantagem uma vez que a sua instalação não é cómoda do ponto de vista do utilizador. Então surgiram novos métodos baseados em TLS, que de algum modo, apaziguaram alguns destes requisitos do EAP-TLS, tais como, o EAP-TTLS e o PEAP. Num contexto

prático apenas estes três métodos são considerados seguros para redes sem fios [7]. O EAP -TTLS e o PEAP são no global mais utilizados por não obrigarem à presença de uma infraestrutura PKI. O EAP-TTLS é mais utilizado e existe uma grande quantidade de produtos que o suportam, o que não se sucede com o PEAP. Outros métodos como EAP-MD5 ou EAP-MSCHAP são vistos como não aconselhados devido a vulnerabilidades de segurança que apresentam.

A adaptação do IEEE 802.1x a redes sem fios deu aso a que novas vulnerabilidades fossem identificadas, por exemplo, a possibilidade de aparecer um AP que não é quem diz ser. Este último aspeto originou novos requisitos operacionais do protocolo, tais como, a autenticação mútua.

2.2. REMOTE AUTHENTICATION DIAL IN USER SERVICE (RADIUS)

Os utilizadores de um *hotspot* baseado em *captive portal*, assim que, se ligarem à rede, são confrontados com uma página de login que, por sua vez, irá desencadear a verificação das credenciais do utilizador. Se este for autenticado com êxito, o *captive portal* irá reconfigurar-se para permitir que o tráfego desse utilizador aceda à Internet.

O *Remote Authentication Dial In User Service* (RADIUS) é um protocolo AAA (Authentication, Authorization and Accounting) que auxilia no controlo do acesso à rede e mobilidade IP.

Tendo em consideração o RFC 2865[14] que especifica o RADIUS, cada cliente envia um pedido ao servidor de acesso à rede, o *Network Access Server* (NAS)[15] para que lhe seja possível aceder aos recursos da rede, por intermédio das credenciais de acesso. Estas são passadas ao dispositivo NAS pela camada de ligação de dados, recorrendo ao protocolo *Point to Point Protocol* (PPP). Desta forma, o NAS envia ao RADIUS uma mensagem de pedido, solicitando a autorização. O pedido inclui credenciais de acesso do utilizador (nome e palavra-chave) ou um certificado fornecido pelo utilizador [16].

O servidor RADIUS confirma se a informação é coerente através de esquemas de autenticação tais como: *Password Authentication Protocol (*PAP) [17], *Challenge Handshake Authentication Protocol* (CHAP) [18] e *Extensible Authentication Protocol* (EAP) [5]. A identidade do utilizador é verificada assim como outras informações tais

como endereço IP do utilizador na rede, estado da conta e os privilégios no acesso a serviços.

O servidor RADIUS retorna para o NAS uma de três respostas possíveis:

- *Access Reject*, em que o acesso aos recursos da rede solicitados pelo utilizador é negado. (ex: falha de confirmação da identidade fornecida)

- *Access Challenge*, em que é pedida ao utilizador informação adicional, como por exemplo, uma chave extra. É bastante utilizado em protocolos de autenticação mais complexos, onde um túnel cifrado é criado entre a máquina do utilizador e o servidor RADIUS para que as credenciais fiquem protegidas.

- *Acess Accept*, que corresponde, à aceitação do pedido do utilizador. Após autenticação o servidor RADIUS verifica periodicamente se o utilizador se encontra autorizado a usar os serviços de rede pretendidos.

Quando o acesso à Internet é concedido ao utilizador pelo NAS, um pedido de contabilização (*Accounting Start*) é enviado pelo NAS para o servidor RADIUS para sinalizar o início da sessão do utilizador na rede. De seguida, é então armazenada a identidade do cliente, endereço IP, ponto de acesso e um identificador de acesso único. Com uma certa cadência é enviada informação atualizada do estado da sessão do cliente pelo NAS para o servidor RADIUS. Caso o acesso seja terminado, o NAS envia registo final de término de sessão (*Accounting Stop*) para o servidor RADIUS, indicando informações como duração da sessão, pacotes, motivo do término da ligação e outras informações relacionadas com ligação à rede por parte do cliente [19].

A Figura 2 apresenta a arquitetura base do protocolo RADIUS.

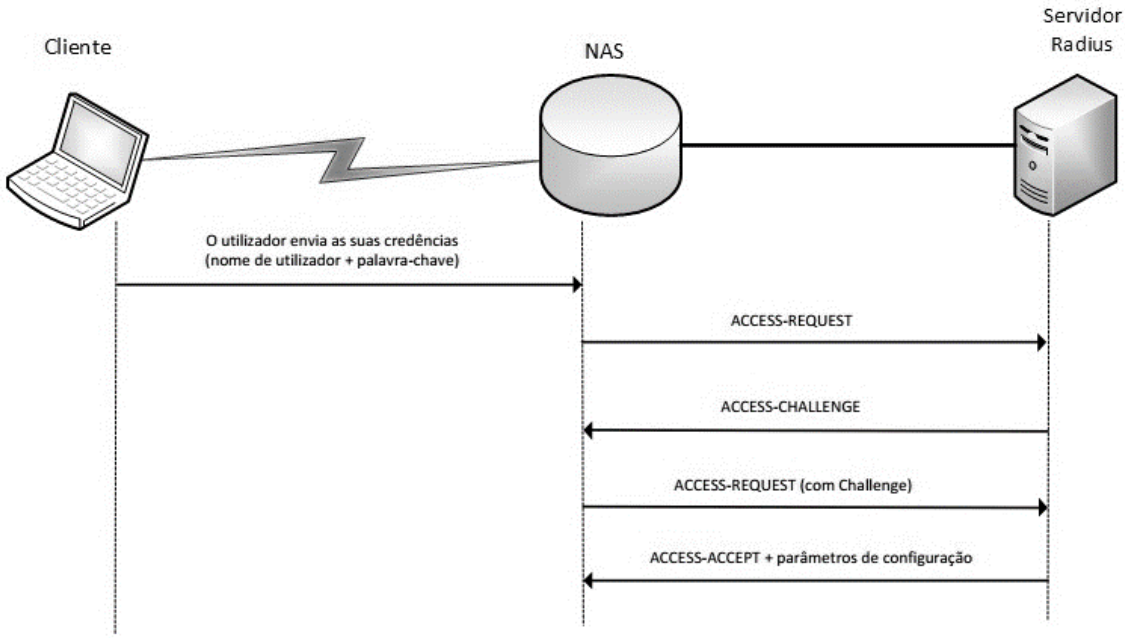


Figura 2 – Representação do protocolo RADIUS

2.3. AUTENTICAÇÃO EM CAPTIVE PORTAL

Cenários relativos à utilização de *captive portals* são comuns em todo o mundo. Fon (*Fon Wireless Ltd.)* é um exemplo de particular interesse, uma vez que está disponível em vários locais. FON consiste numa comunidade que liga *hotspots wireless,* fornecidos pelos utilizadores, que foi fundada em Madrid [20]. Construiu a sua comunidade com a venda de *routers* personalizados ("*La Fonera*") que autentica os utilizadores através da ligação a um servidor Fon, por intermédio da Internet. Cada *router* inclui um ponto de acesso que anuncia duas redes sem fio: uma rede criptografada para o proprietário do mesmo, e uma rede aberta ao público para outros utilizadores Fon que opera um *captive portal.*

A Fon possui membros passivos e ativos. Os membros passivos não compartilham a sua ligação de Internet com a comunidade e, se estes pretenderem utilizar pontos Fon, é necessário pagar por isso. Os membros ativos possuem um *router Fonera* e compartilham a sua ligação à Internet com outros membros, em troca de *roaming* gratuito em outros pontos Fon [21]. Considerando-se o caso de utilização Fon, os membros ativos podem ser

vistos como utilizadores e/ou operadores *captive portal*, enquanto Fon irá servir como operador de autenticação.

Os *captive portals* redirecionam um cliente http/https para uma página *Web*, em particular quando este acede a uma página solicitada por si, requisitando a autenticação, pagamento e/ou apresentando as políticas de utilização, normalmente exigindo, que o utilizador as aceite para prosseguir com a navegação. O *captive portal* captura todos os pacotes, não dependendo do endereço IP ou porta, até que o cliente abra um navegador e tente solicitar uma página. A partir desse instante, ele captura todo o tráfego ao nível IP e direciona para a página configurada [16].

Os *captive portals* são bastante comuns em *hotspots* sem fios e podem ser utilizados para controlo de acessos por cabo em quartos de hotéis ou apartamentos.

Em baixo é apresentada uma lista de *software* gratuito que permite configurar *captive portals* no sistema operativo Linux:

- CoovaChilli – é um controlador de acesso *open-source* para *captive portal Universal Access Method* (UAM) e 802.1x baseado no popular ChilliSpot mas agora extinto.
- ChilliSpot – captive portal de código aberto ou controlador de acesso LAN sem fios. Apesar da transição de uma parte da comunidade para o *coovachilli* é possível a sua instalação.
- EasyHotspot – software *open-source* escrito em *PHP* e *MySQL*.
- NoCat – desenvolvido pelo NoCat escrito em Perl, captura o tráfego da porta 80 e desvia para o servidor http seguro. Não possui ferramenta de interface Web, sistema de autorização e existe falta de documentação oficial.
- WifiDog – desenvolvido em C para torná-lo fácil de incluir em sistemas embebidos.

2.3.1. SERVIDOR PPPOE

O protocolo ponto a ponto (PPP) tem como objetivo transportar o tráfego entre dois dispositivos de rede através de uma só ligação física. *Point to Point Protocol over Ethernet* (PPPoE), é um protocolo da camada de ligação de dados que utiliza o PPP para ligar o cliente a um servidor através de uma ligação ponto a ponto (RFC 2516[22]).

Uma importante funcionalidade é que todo o tráfego, que tem como destino um determinado cliente ligado por PPP, terá de passar necessariamente pelo servidor PPPoE para poder chegar ao cliente. Um servidor PPPoE assume-se como ferramenta para encaminhamento de pacotes, NAT, *firewall* e controlo de tráfego. O pacote *pppd* para sistemas Linux suporta parâmetros RADIUS para autenticação de clientes [16].

2.3.2. ACCESS POINT

Access Point (AP) ou ponto de acesso é um dispositivo numa rede sem fios que interliga todos os dispositivos móveis. Estes fornecem acesso sem fios a uma rede Ethernet com fios.

Um ponto de acesso é ligado a um concentrador, comutador ou *router* com fios e emite sinais pelo meio. Tornando possível a ligação de computadores e dispositivos de uma rede sem fios a uma rede com fios. Os pontos de acesso, suportam *handover*, assim o utilizador pode mover-se de uma localização para outra e continuar a ter acesso sem fios a uma rede. O *handover* ocorre quando um dispositivo móvel sai da área de cobertura de uma célula e entra numa célula vizinha.

Habitualmente está ligado a uma rede por cabo servindo de ponto de acesso para outra rede, como por exemplo a Internet. Um dispositivo ao ligar-se à Internet por uma rede sem fios através da utilização de uma rede pública sem fios, como é o caso de um aeroporto, café ou hotel está habitualmente a ligar-se a um ponto de acesso. Se se pretender ligar um computador a uma rede sem fios e se tiver um *router* que forneça sinal sem fios, não é necessário um ponto de acesso. [23]

2.3.3. SUPPLICANT

O suplicante (*Supplicant*) é um *software* cliente que comunica com o operador de autenticação por intermédio de um ponto de acesso para realizar autenticação (servidor RADIUS).

Para se ligar a uma rede 802.1x, deve-se instalar o suplicante (cliente) na máquina do utilizador. Num passado relativamente próximo, não era fácil a tarefa de instalar o cliente para utilização em Linux. No entanto, hoje em dia, é diferente pois, grande parte das distribuições Linux possui integradas aplicações para as configurações 802.1x, que simplifica a configuração e introdução das credenciais.

Os dois principais projetos relativos a suplicantes 802.1x em Linux são Xsupplicant [24] e wpa_supplicant[25]. O Xsupplicant existe desde 2003 e é desenvolvido por Open1X e desenvolvido pela OpenSEA Alliance. O wpa_supplicant existe desde 2004 e é desenvolvido por Jouni Malinen e outros colaboradores. Ambos os clientes podem ser executados em Linux e Windows e tem uma aplicação gráfica, além de configuração baseada em texto. O projeto wpa_supplicant também suporta BSD e Mac OS X.

2.4. FUNCIONAMENTO GERAL

Hotspots wireless baseados em *captive portal* operam, capturando todo o tráfego do utilizador até que a sua autenticação seja bem-sucedida. A autenticação do utilizador é feita através de uma página *Web*. A Figura 3 representa as interações existentes na ligação de um utilizador e na sua autenticação ao aceder à Internet através de um *captive portal*.

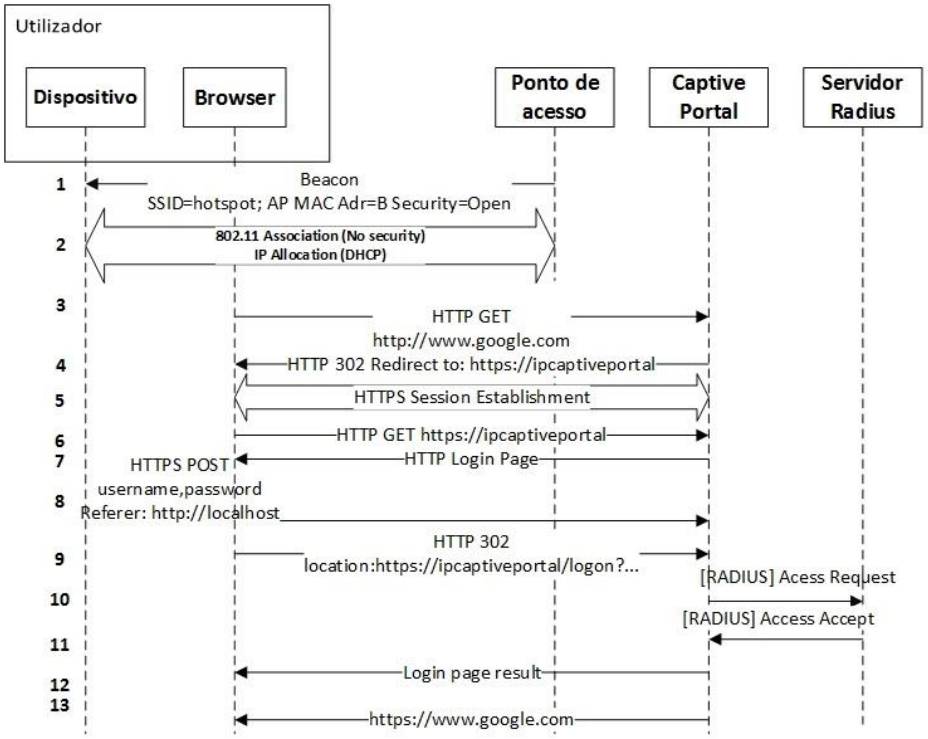


Figura 3 – Representação do funcionamento do captive portal.

O ponto de acesso (AP) transmite periodicamente em *broadcast* o *beacon* contendo SSID dos *hotspots*, o endereço MAC do AP e as informações de segurança. A informação de segurança está definida como aberta, ou nenhuma (passo 1 da Figura 3). Em seguida, o dispositivo de conexão obtém a sua configuração IP (*Dynamic Host Configuration Protocol* (DHCP)) e permite que o utilizador comece a sua navegação na *Web* (passo 2*)*. Todas as comunicações de IP de utilizadores desconhecidos são bloqueadas, exceto para DHCP, DNS e protocolo *Address Resolution Protocol* (ARP).

Neste exemplo, o utilizador ao tentar aceder (passo 3) a um *website* (www.google.com) é redirecionado para uma ligação HTTPS para uma página inicial pré-configurada do *captive portal*. A mensagem HTTP 302 é utilizada para redireccionamento (passo 4). A sessão HTTPS é criada entre o cliente e o *captive portal* para garantir a entrega segura de dados do utilizador (passo 5). Desta forma, o cliente envia uma mensagem HTTP GET (passo 6) para o URL compreendida no campo de localização da mensagem recebida HTTP 302. Este URL contém o endereço da página de *login* do *captive portal*.

O utilizador digita as suas credenciais (nome de utilizador e senha) na página de login (passo 7). Estas são enviadas ao *captive portal* através de uma mensagem POST HTTPS (passo 8). Ao receber as credenciais do utilizador (passo 9), ele encaminha para o servidor RADIUS para autenticação do utilizador.

O servidor RADIUS, com base nestas informações, decide se a autenticação deve ter sucesso ou falhar e responde de volta para o *captive portal.* Após a autenticação bem-sucedida (passos 10 e 11), o *captive portal* remove a regra de redireccionamento HTTP aplicado a esse cliente em específico. O captive portal informa ao utilizador o resultado da autenticação (passo 12). O utilizador terá, então, permissão para aceder à Internet (passo 13).

3. CRIPTOGRAFIA APLICADA

A criptografia é uma técnica utilizada para proteger informação. A mensagem original é vista como um conjunto de símbolos que se pretende manter sigiloso. O resultado da operação de cifragem da mensagem original denomina-se de criptograma.

A informação da mensagem durante a transmissão fica incompreensível, exceto para o recetor e emissor da informação, que tem a possibilidade de reverter o processo. A cifragem permite a transformação reversível da informação, de modo a torná-la impercetível; e a decifragem é a sua operação inversa. Uma cifra é um algoritmo criptográfico, função matemática injetiva, que efetua as transformações entre o texto limpo e o criptograma [26]. De salientar que, quanto maior for número de bits que a chave (tamanho) de cifragem apresenta, mais moroso é de descobrir por método de força bruta.

Preliminarmente, as técnicas criptográficas estariam associadas a atividades militares e diplomáticas. Os avanços da ciência permitiram o desenvolvimento de técnicas mais avançadas progredindo-se da criptografia clássica para a criptografia moderna [27].

As principais propriedades da criptografia são a confidencialidade (garantindo que apenas quem se pretende tem a possibilidade de ler os dados), a autenticação (garantindo

que a informação tem origem fidedigna) e integridade (garantindo que a informação não foi alterada entre o emissor e o recetor) [28]. Para além dos casos em que se pode aplicar, já enunciados, existem outros como é, por exemplo, o caso da assinatura digital.

Na prática, ao mesmo tempo que se utilizam os algoritmos utilizam-se também chaves, pois mesmo que o algoritmo seja conhecido é necessário possuir a chave correta para decifrar a informação. Não existem mecanismos de cifragem/decifragem com eficácia de 100%, no contexto teórico facilmente se pode pensar que qualquer chave pode ser quebrada por força bruta, isto é, tendo em conta que se possui um exemplar de uma mensagem original e cifrada, e o algoritmo é conhecido, podendo-se então tentar todas as chaves possíveis até que uma coincida.

Apesar disto no contexto prático, e tendo em consideração as capacidades de processamento do equipamento atual, a solução é utilizar algoritmos e chaves que não possam ser descobertas em tempo útil. O tempo para que uma chave descoberta por força bruta seja alcançada depende do número de chaves possíveis, do seu número de bits e do tempo de execução do algoritmo.

O senão desta abordagem é que segundo Gordon Earl Moore, fundador da Intel, "a capacidade de processamento dos computadores duplica em cada 18 meses" sendo esta lei conhecida como lei de Moore [29]. Portanto a cada 18 meses deverá ser necessário aumentar um bit às chaves utilizadas.

3.1. CRIPTOGRAFIA SIMÉTRICA

A criptografia simétrica também conhecida como criptografia tradicional ou de chave secreta utiliza a mesma chave para as operações de cifragem e decifragem [30].

A mensagem inteligível original, referida como mensagem corrente é transformada em mensagem ilegível, a mensagem cifrada. O processo de cifragem tem por base um algoritmo e uma chave. O algoritmo produz uma saída diferente em conformidade com a chave utilizada no momento. Assim que a mensagem é cifrada é transmitida. Na receção é convertida na mensagem original utilizando o algoritmo de decifragem e a mesma chave que foi utilizada na cifragem.

Este tipo de segurança depende de diversos fatores. O algoritmo deve ser suficientemente poderoso para que não seja possível decifrar uma mensagem com base

unicamente na mensagem cifrada. Como é notório, é necessário manter as chaves secretas para que seja mantida a confidencialidade da mensagem. A segurança depende do secretismo da chave e não do algoritmo utilizado. O facto de o algoritmo não necessitar de ser mantido em sigilo, permite aos fabricantes de *chips* poderem desenvolvê-los a menor custo. Assim, o principal problema é manter a chave como secreta.

Os algoritmos que utilizam este tipo de criptografia são tendencialmente mais rápidos, no entanto não são considerados tão seguros como os da criptografia assimétrica.

Habitualmente são distinguidas duas classes fundamentais no que se refere à criptografia simétrica: com cifra por blocos e com cifra sequencial.

3.1.1. CIFRA POR BLOCOS

Este método divide a mensagem de entrada em blocos de tamanho fixo (habitualmente 64 bits) e cifra um bloco de cada vez. No caso do tamanho da mensagem original não ser múltiplo do tamanho do bloco, o último bloco é completado tendo em conta um algoritmo de *padding*. São cifras fundamentais aos sistemas criptográficos uma vez que as principais técnicas são deste tipo[31]. Alguns exemplos são:

- *Data Encryption Standard* (DES)

Este algoritmo é uma cifra simétrica por blocos, desenvolvido pela IBM e mais tarde adotado como *standard* pelo *National Institute of Standards and Technology* (NIST). Este cifra blocos de 64 bits em blocos de 64 bits utilizando uma chave de 64 bits (56 bits de dados e 8 bits de paridade). Assim, na cifragem de uma mensagem escolhe-se uma chave e divide-se a mensagem em blocos de 64 bits, sendo cada bloco cifrado de forma separada. O algoritmo DES integra a aplicação sucessiva, repetida 16 vezes, de cifra em bloco, denominado de *round*. Para cada *round* é derivada uma chave (K) de *round* através de uma permuta de chave da cifra (*key shedule*). Um *round* do DES é construído com base no circuito de Feistel, permitindo implementações eficientes. O processo de cifragem difere do processo decifragem, por ser a operação inversa e na ordem de aplicação das chaves que é também inversa [32].

A Figura 4 representa a estrutura do circuito de Feistel, onde cada bloco de texto limpo é dividido em duas partes de 32 bits (D e E). Um aparece sem modificações no final do *round*, sendo que são necessários dois rounds para cifrar todos os bits do bloco. A outra

parte que é cifrada passa pela função F (Feistel), onde é permutada, combinada com a chave de *round* e passada por uma função não linear [32].

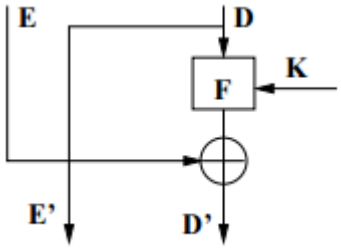


Figura 4 – Representação do circuito de Feistel. [26]

Hoje em dia a sua utilização não é aconselhável visto ser vulnerável a ataques de força bruta (reduzida dimensão das chaves). Este algoritmo foi bastante utilizado em aplicações bancárias.

- *Triple Data Encryption Standard* (3DES)

O *Triple Data Encryption Standard* é um algoritmo de cifra simétrica por blocos tendo por base o DES (sequência cifragem-decifragem-cifragem com três chaves). Apresenta uma cifra em cascata que permite utilizar o DES com chaves de 56, 112 ou 168 bits úteis mediante as chaves C1,C2,C3 sejam iguais ou diferentes entre si, no processo de cifragem em cascata (Figura 5).

Mesmo que considerado bastante mais seguro do que o DES, é mais lento quando comparado com as alternativas atuais. Com o 3DES é possível a sua utilização em micro controladores com capacidade de processamento e memória limitadas [32].

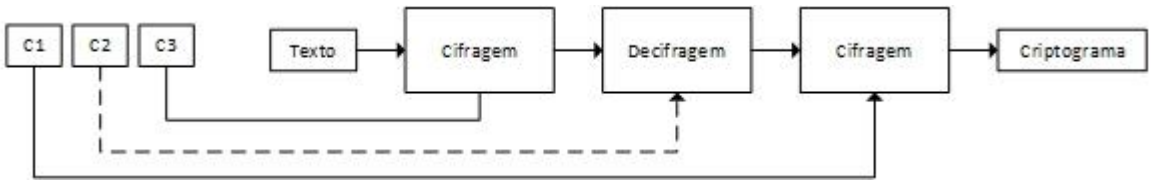


Figura 5 – Representação da cifra em cascata.

- *International Data Encryption Algorithm* (IDEA)

O *International Data Encryption Algorithm (IDEA)* é um algoritmo simétrico por blocos desenvolvido em 1990, e para a sua utilização é necessária uma licença do proprietário a ASCOM. Neste algoritmo o tamanho de cada bloco é de 64 bits, a chave é de

128 bits e as operações de cifragem e decifragem utilizam o mesmo algoritmo permitindo implementações eficientes. Tem por base a partição do bloco em quatro partes e na aplicação contínua de operações algébricas [26].

- *Blowfish*

Algoritmo simétrico por blocos desenvolvido por Bruce Schneier. Apresenta blocos com tamanho de 64 bits e chaves de tamanho variável (até 448 bits). Este algoritmo adquiriu grande popularidade sendo uma alternativa gratuita e eficiente aos algoritmos existentes. O *Blowfish* foi aceite, sendo particularmente utilizado em aplicações como Nautilus e PGPfone [27].

Os ataques de conhecimento público têm por base a classe de chaves ser fraca. Algumas das vantagens do *Blowfish* são ser considerado: mais rápido que o DES e o IDEA, não ser patenteado, não necessitar de licença e disponibilizar o código fonte [27].

- *Rivest Cypher* 5 (RC5)

Apresenta-se como sendo uma técnica de cifragem em bloco, é caracterizado por uma grande flexibilidade e possibilidade de parametrização. Os blocos de entrada podem ter dimensão pré-determinada, a chave é de comprimento variável e é possível determinar o número de interações do algoritmo *Rivest Cypher 5.* Por este apresentar flexibilidade, uma série de parâmetros podem ser ajustados em conformidade com a utilização pretendida. A mensagem original é fornecida ao algoritmo sob forma de dois blocos n bits, sendo que mediante a situação, n pode tomar valores de 16, 32 ou 64 [33].

Este algoritmo é considerado: apropriado para ser implementado em *hardware* e *software*, rápido e simples com necessidade baixa de memória. Um documento popular de conhecimento público é "*The RC5 Encryption Algorithm*"[34] que revela detalhes sobre o RC5.

- *Advanced Encryption Standard* (AES)

O *Advanced Encryption Standard* (AES) é um algoritmo de cifragem simétrico por blocos, também conhecido por Rijndael. Foi desenvolvido por uma equipa de investigadores de uma universidade belga e foi o vencedor do concurso para o padrão, tendo sido anunciado pela NIST em 26 de Novembro de 2001 sendo padrão em 2002.

No AES, o emissor e recetor utilizam uma chave única para cifrar e decifrar, o comprimento de bloco de dados tem comprimento de 128 bits, o comprimento da chave é variável, podendo tomar valor de 128, 192 ou 256 bits. É um algoritmo iterativo e o número de *rounds* pode ser 10, 12 ou 14 com o respetivo comprimento de chave 128, 192 ou 256 bits. Cada round no AES, com a exceção do último, passa por 4 fases que são: *SubBytes, ShiftRows, MixColumns* e *AddRoundKey* [32].

A fase SubByte é uma substituição não linear que opera em cada byte da matriz de estado independente (matriz bidimensional, no caso do AES-128, 4x4=16 bytes). Esta é a operação mais complexa do AES e garante que a não-linearidade é introduzida durante a cifragem. Estes elementos são definidos como *SBoxes*.

Na fase *ShiftRows*, as linhas da matriz de estado são alteradas (deslocadas para a direita) ciclicamente sendo C1, C2 e C3 o deslocamento que se deve aplicar nas linhas da matriz de estado. A primeira linha não é alterada, a segunda linha é deslocada em C1 bytes, a terceira linha é deslocada em C2 bytes e a quarta linha é deslocada em C3 bytes. Os parâmetros C1, C2 e C3 são dependentes do número de bytes da coluna.

Na fase *MixColumns*, cada coluna é processada através de multiplicação de vetores em campo binário. É a principal operação de difusão do AES, e não é calculado no *round* final. A fase *AddRoundKey*, consiste somente em aplicar a operação lógica (XOR) na matriz de estado, usando a matriz proveniente da função de escalonamento de chaves. No *AddRoundKey* o XOR, consiste em, bit a bit do bloco atual com uma porção da chave expandida, ou seja, os 128 bits da matriz de estado são utilizados num XOR bit a bit com os 128 bits da chave de *round*.

O processo de decifragem é o inverso do de cifragem.

O AES é considerado vulnerável a ataques de chaves relacionadas, quando se utiliza uma chave que provém da chave anterior, o que não é reprovável desde que se utilizem sempre chaves variáveis.

Como habitual das cifras por blocos, a otimização da sua eficiência depende da utilização de *SBoxes*, o que aumenta a sua vulnerabilidade a *side channel attacks*, em que se procura recolher informação que permita descobrir a chave. Não obstante de uma chave de 128 bits ser considerada segura, é aconselhável a utilização de chaves de 256 bits.

O algoritmo AES foi desenvolvido com o intuito de resistir aos ataques de descodificação, como os métodos de análise de tempo e análise de potência. De modo a proporcionar ao utilizador ainda mais benefícios, AES, quando idealmente aplicado necessita de pouca memória para codificar e descodificar.

O algoritmo AES revela particular interesse na indústria aeroespacial, incluindo satélites devido à necessidade de proteção dos dados transmitidos desde os satélites para o solo [35].

O algoritmo AES foi projetado tendo em consideração as seguintes características: algoritmo público, resistência contra ataques, velocidade, implementação eficiente em *hardware* e *software* (Java e C) em três tipos de plataformas (32 bits, 64 bits e *smartcards*).

3.1.2. CIFRA SEQUENCIAL

É vista como cifra de blocos em que o tamanho do bloco é 1. Opera sobre fluxos de tempo em aberto, um bit ou byte de cada vez, combinando-o com um fluxo de chaves gerado de forma interna.[31] O esquema de decifragem é semelhante ao de cifragem (Figura 6). A sequência de chaves tem de ser reproduzida de forma exata. A chave da cifra é a semente do gerador de chaves. A segurança da cifra advém da dificuldade de antever a sequência de valores gerados. As cifras sequenciais podem ser classificadas em síncronas e auto-sincronizáveis[26].

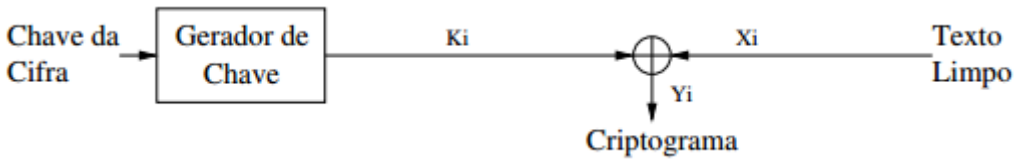


Figura 6 - Representação do esquema de cifragem de cifra sequencial [26].

No que se refere às primeiras, o fluxo de chaves é gerado de forma independente dos dados cifrados. Este modo é vigoroso visto que não ocorre propagação de erros quando os bits são recebidos de modo errado. No entanto, um atacante que saiba os bits que pretende alterar pode não ser descoberto.

Quanto às cifras auto-sincronizáveis, o fluxo de chaves é determinado com base no criptograma que foi gerado, o que possibilita recuperar perdas de sincronismo. O problema

destas cifras é que propagam os erros que acontecem no criptograma, apesar de serem menos vulneráveis a ataques de repetição.

Um exemplo de cifra sequencial é o algoritmo *Rivest Cypher* 4 (RC4) criado por Ronald Rivest para a empresa RSA. Ao início foi mantido secreto mas em 1994 é tornado público. O RC4 é utilizado na cifragem de protocolos como *Secure Sockets Layer* (SSL) e WEP, apesar de não ser considerado seguro. Permite chaves entre os 40 e 2048 bits. Esta técnica é considerada 4 vezes mais rápida do que o DES simples. Outro exemplo o é *Software-Optimized Encryption Algorithm* (SEAL) que deriva de técnicas utilizadas em funções de *hash: Secure Hash Algorithm* (SHA) e SHA-1, utilizado na cifra de discos rígidos.

3.1.3. MODOS DE CIFRA

Os critérios para a escolha de cada uma das variantes são: a segurança, a eficiência, a propagação/tolerância a erros e o desempenho adequado do modo de funcionamento.

Os modos mais comuns são [31]:

- Modo ECB (*Electronic Code Book*)

A Figura 7 representa o esquema de cifragem do modo ECB. Neste modo, caso a chave, o bloco de entrada e de saída tiverem todos *n* bits, a cifra de bloco define o mapeamento de um para um dos *n* bits inteiros, tendo em conta, permutações de *n* bits inteiros. No caso de o mesmo bloco ser cifrado duas vezes com a chave igual, o resultado do bloco de texto cifrado será também igual. Esta informação é particularmente útil para o atacante, uma vez que o texto em claro duplicado pode expor a chave, e então ser possível que todos os blocos sejam decifrados. Exemplo: implementação simples do DES.

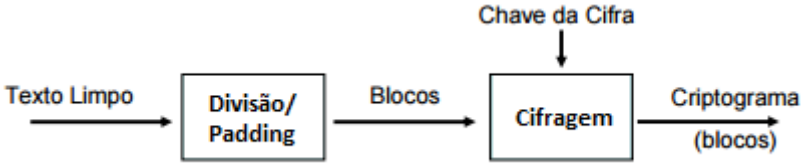


Figura 7 - Representação do esquema de cifragem do modo ECB.[31].

- Modo CBC (*Cipher-Block Chaining*)

A Figura 8 representa o esquema de cifragem do modo CBC. O bloco de texto cifrado é concebido pela operação de XOR do bloco de texto simples com o bloco cifrado anterior, cifrando o valor resultante. Ao bloco inicial é adicionado, na operação XOR um vetor de inicialização (*Initialization Vector* (IV)). Assim, os primeiros blocos influenciam os restantes, o que aumenta o número de bits do texto simples de que um bit de texto cifrado depende e pode originar problemas de sincronização aquando a perda de um bloco. Este modo aperfeiçoa a segurança do DES no que se refere à pesquisa de chaves.

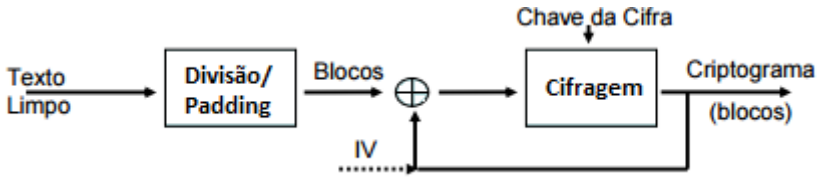


Figura 8 - Representação do esquema de cifragem do modo CBC.[31]

- Modo CFB (*Cipher Feedback*)

A Figura 9 representa o esquema de cifragem do modo CFB. O bloco de texto cifrado na posição *i*, é obtido pela cifragem do bloco na posição (*i*-1) e pela operação XOR do seu resultado no texto simples. O *shift-register* no início é preenchido com o vetor de inicialização (IV) que é utilizado como "semente" na operação. Apresenta como vantagem não ter necessidade de *padding*.

A necessidade de *padding* prende-se com o facto das cifras de blocos poderem apenas cifrar conjuntos de *n* bits. Portanto, sempre que o texto original não é múltiplo de *n*, é então necessário preencher os bits que restam com um esquema de *padding*.

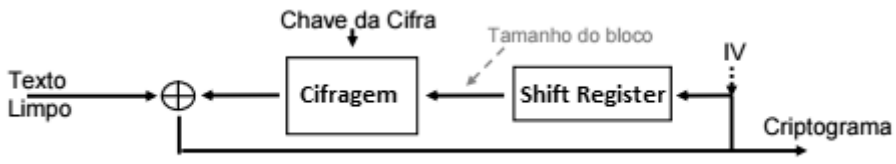


Figura 9 - Representação do esquema de cifragem do modo CFB[31].

- Modo OFB (*Output Feedback*)

A Figura 10 representa o esquema de cifragem do modo OFB, que corresponde a uma cifra sequência síncrona. O OFB é idêntico ao CFB à exceção de que os blocos adicionados na operação XOR com cada bloco de texto em claro é gerado independentemente dos blocos de texto em claro e do texto cifrado. O *shift-register* e IV têm a cargo a mesma função que no CFB mas o *shift-register* é alimentado pelos bits da chave já utilizados.

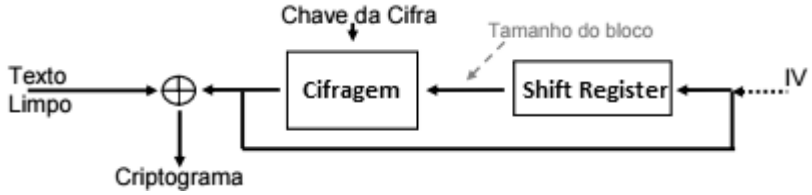


Figura 10 - Representação do esquema de cifragem do modo OFB.[31]

3.2. CRIPTOGRAFIA ASSIMÉTRICA

A criptografia assimétrica ou de chave pública é normalmente utilizada para garantir sigilo ou para implementação de assinaturas digitais.[32] Utiliza o conceito de chaves públicas e privadas em que cada utilizador possui uma chave pública e uma privada. A chave privada mantém-se com o utilizador, unicamente a chave pública pode ser distribuída [31].

Apesar da chave de decifragem necessitar de ser secreta (chave privada), só o recetor tem necessidade de a conhecer. Um atacante pode conhecer a chave de cifragem, sem que isso comprometa a segurança da cifra. Embora as cifras assimétricas possam ser mais seguras que as cifras simétricas, tendo em conta que nas cifras simétricas é mais complexa a gestão e distribuição das chaves, salienta-se que na realidade as cifras assimétricas não são um substituto para as cifras simétricas, uma vez que as primeiras são mais lentas para o mesmo nível de segurança [36].

Um algoritmo de criptografia assimétrica extensamente utilizado é o *Rivest, Shamir e Adleman* (RSA). A operação de cifragem é baseada no problema de fatorizar um número inteiro grande nos seus fatores primos. São conhecidos ataques, relativos à factorização de inteiros, já concretizados para 768 bits. No entanto para chaves de pelo menos de 1024 bits é considerada segura a implementação.

Quanto maiores as chaves mais difícil é a sua descoberta, sendo que, a sua utilização prolonga-se no tempo, no entanto, as operações de criptografia são também mais lentas.

A Tabela 1 representa a comparação entre as dimensões das chaves dos algoritmos de chaves simétricas e assimétricas. Quanto ao tamanho do par de chaves é recomendável que seja selecionado de acordo com o período pelo qual se pretende assegurar a confidencialidade dos dados cifrados. Podem ser descritos três intervalos de forma resumida tendo em conta o tamanho mínimo da chave: a curto prazo utilização de 1024 bits, até 2020 utilização de 2048 bits e após 2030 o tamanho mínimo previsto é 4096 bits [37]. No que se refere ao tamanho das chaves utilizadas para assinatura, este não é tão crítico, caso sejam alteradas com frequência, e seja pedida a aposição de selos temporais às assinaturas efetuadas.

Tabela 1 – Força criptográfica de chaves simétricas e assimétricas. [38]

Chave Simétrica (bits)	Algoritmo de criptografia Simétrica	Chave assimétrica RSA (bits)
80	Skipjack	1024
112	3DES	2048
128	AES-128	3072
192	AES-192	7680
256	AES-256	15360

3.3. FUNÇÕES DE HASH

As funções de *hash* criptográficas (também chamadas de funções de resumo, *one-way*, impressões digitais, etc) são práticas recorrentes em vários protocolos criptográficos de forma a garantir a sua integridade. Uma vez que a informação adicional que os

mecanismos oferecem pode ser alterada, a garantia de integridade é conseguida através da conjugação destes mecanismos com os mecanismos de assinatura digital [26].

As funções de *hash* são funções ditas unidirecionais que transformam uma entrada de tamanho variável numa saída de tamanho fixo [39]. Para garantir a integridade importa extrair uma impressão digital não invertível de uma mensagem obtendo-se um valor que identifique o conteúdo dessa mensagem. Como as funções de *hash* não são injetivas, as mensagens poderão ser mapeadas no mesmo valor de hash [26].

No entanto, o processo de identificação recorre a probabilidades, sendo considerado se é ou não provável que o valor de *hash* tenha sido originado por aquela mensagem. Para tal são utilizadas funções de *hash* definidas como livres de colisões (para mensagens diferentes é originado o mesmo valor de *hash*) em que, além de não haver uma relação visível entre a entrada e a saída, é muito difícil encontrar duas mensagens que originem o mesmo valor de *hash* (fortemente livre de colisões). Dado um valor de *hash* e a mensagem que o originou, é árduo arranjar outra mensagem que origine o mesmo valor de *hash* (fracamente livre de colisões). As funções de *hash* deste tipo podem ser tornadas públicas. Existe baixa probabilidade de encontrar duas mensagens com o mesmo valor de *hash* não comprometendo a sua segurança [26].

Um *Message Authentication Code* (MAC) pode ser considerado como uma função de *hash* cujo resultado está dependente, tanto da mensagem como da chave secreta.

Os algoritmos de *hash* mais populares são *Message Digest 5* (MD5) e *Standard Hash Algorith* (SHA). Quanto ao MD5, foi desenvolvido por Ron Rivest, tem como entrada uma mensagem de tamanho arbitrário e produz na saída uma mensagem resumida de 128 bits. A entrada é processada em blocos de 512 bits[40]. No contexto atual, não é aconselhável a sua utilização devido ao comprimento reduzido da mensagem resumo e de terem sido descobertas vulnerabilidades de segurança que permitem encontrar duas mensagens com o mesmo valor da função de resumo [26].

No que se refere ao SHA, a versão SHA-1, publicada em 1995 pelo NIST produz um valor de *hash* de 160 bits. Não é aconselhável a sua utilização devido às vulnerabilidades encontradas que reportam a possibilidade de encontrar duas mensagens com mesmo valor da função de resumo. No entanto em 2002, é publicada a versão SHA-2 que permite produzir uma função de resumo de 224,256, 384 e 512 bits [40].

3.4. ASSINATURA DIGITAL

Definida pelos autores em [41] como a "Modalidade de assinatura eletrónica avançada baseada em sistema criptográfico assimétrico composto de um algoritmo ou série de algoritmos, mediante o qual é gerado um par de chaves assimétricas exclusivas e interdependentes, uma das quais privada e outra pública, e que permite ao titular usar a chave privada para declarar a autoria do documento eletrónico ao qual a assinatura é aposta e concordância com o seu conteúdo e ao destinatário usar a chave pública para verificar se a assinatura foi criada mediante o uso da correspondente chave privada e se o documento eletrónico foi alterado depois de aposta a assinatura."

Permite que se verifique a integridade da informação e pode ser utilizada para a confirmação que o emissor de uma mensagem é verdadeiro.

A assinatura deve ser [26]:

- Autêntica: convence o recetor do documento de que o remetente explicitamente assinou o documento.
- Não falsificável: prova que o remetente, e não outra pessoa, assinou o documento.
- Não reutilizável: faz parte do documento e não se pode transpor para outro documento.
- Garantia da integridade do documento: o documento permaneceu inalterado desde que foi assinado.
- Não repudiável: o remetente não pode depois negar que assinou o documento.

Pode ser considerado como o principal contributo da criptografia assimétrica, o facto de permitir a um documento digital o conceito de assinatura de um documento [42]. O caso mais conhecido é o algoritmo RSA.

4. SECUP – MIGRAÇÃO DE REDES WIFI

Tendo em consideração a implementação de um *captive portal*, para o controlo e gestão de acessos dos utilizadores e um servidor RADIUS para a validação das credenciais é apresentada a proposta de solução.

O *captive portal* é utilizado para autenticar os clientes presentes numa LAN sem fios ou com fios e gerir alocação dos endereços IP com o auxílio do protocolo DHCP.

A validação das credenciais dos utilizadores é feita recorrendo a uma página *Web*, em que servidor RADIUS fica responsável pela autenticação e contabilização.

O *captive portal* suporta autenticação UAM (*Universal Access Method*) e WPA. Relativamente ao UAM o cliente solicita um endereço IP ao servidor, e é atribuído um endereço IP. Quando o cliente abre um navegador *Web*, o *captive portal* captura o tráfego da ligação e redireciona para o servidor *Web* de autenticação. O servidor *Web* solicita a identificação do utilizador e senha ao cliente, esta é cifrada e enviada para o *captive portal* [16].

No caso do WPA a autenticação é resolvida no *hotspot* sem fios, e de seguida reencaminhada para o *captive portal* no servidor. Quer para o WPA ou UAM, o *captive portal* redireciona o pedido de autenticação para o servidor RADIUS. O RADIUS envia uma mensagem de aceitação para o *captive portal* se as credenciais forem válidas ou de negação no caso de não serem. A entidade que irá atribuir os endereços IP é o *captive portal*, e para isso, ele cria um dispositivo virtual de rede para *tunnelling* (TUN).

4.1. PROPOSTA DE SOLUÇÃO

A solução proposta mantém o funcionamento global do *captive portal* enquanto permite a troca segura de ficheiros de configuração, para que os clientes possam migrar automaticamente para acesso seguro à rede após a autenticação bem-sucedida.

A solução proposta introduz duas novas aplicações desenvolvidas em java. Recorreu-se à plataforma Java, pois é, uma linguagem multiplataforma, suporta processamento paralelo múltiplo e possuí uma vasta comunidade com apoio documentado.

Quanto ao algoritmo AES que veio substituir o DES, foi utilizado porque é considerado: seguro (resistência a criptoanálise), eficiente a nível computacional e requisitos de memória, implementação simples do algoritmo e apropriado para *software* e *hardware*. É considerado uma cifra por bloco, o que em caso de perda/alteração de bits afeta o bloco em questão (o oposto das cifras sequenciais, que podem afetar a restante informação). É um algoritmo que permite chave variável podendo-se adaptar o nível de criptografia à aplicação necessária de forma comoda (ex: chave 128, 192 e 256 bits). A principal vantagem relativa à escolha da sua implementação é a simplicidade, esta técnica apresenta facilidade de utilização e rapidez para executar os processos criptográficos juntamente com a linguagem Java.

A Figura 11 apresenta o fluxograma relativo ao lado do cliente, descrevendo o funcionamento da aplicação java do lado do cliente, presente no Anexo A. Inicialmente, o programa obtém do utilizador o *username* e respetiva senha a partir da mensagem HTTP redirecionada. Foi instalado e configurado, no servidor, um certificado SSL de modo a estabelecer-se sessão HTTPS. Uma chave é então gerada (*AuthToken*), que será utilizada para autenticação e cifragem do ficheiro. Em seguida, o ficheiro de configuração *wireless* é recebido e decifrado. Numa fase seguinte, a aplicação cliente é executada, usando o

wpa_supplicant o novo ficheiro de configuração para estabelecer uma ligação sem fios segura.

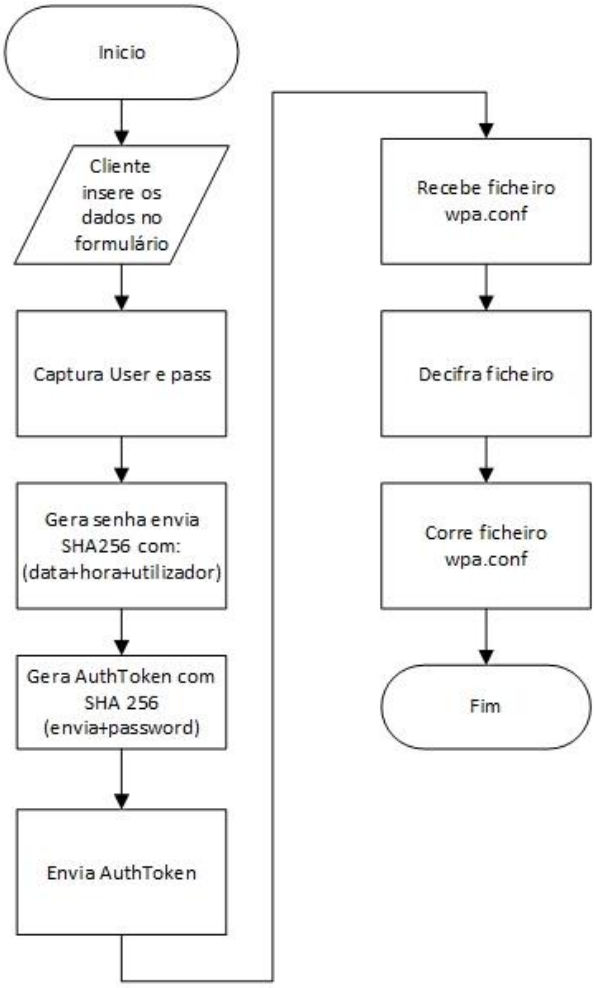


Figura 11 – Fluxograma relativo ao Cliente.

O Código 1, apresenta o ficheiro de configuração que o cliente recebe para que estabeleça autenticação numa sessão WPA. O ficheiro de configuração não é armazenado no servidor, este é escrito aquando a solicitação por parte do cliente. Os dados "*user*" e "*password*" variam consoante o utilizador.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
eapol_version=2
ap_scan=0
network={
        key_mgmt=WPA-EAP
        eap=TTLS
        identity=user
        password=pass
        priority=1
        eapol_flags=0
}
```

Código 1 – Ficheiro de configuração WPA para o suplicante.

A Figura 12 apresenta o fluxograma do funcionamento ao lado do Servidor, descreve o funcionamento da aplicação java do lado do servidor, presente no Anexo B e no Anexo C. No início, a aplicação espera por ligações de clientes. Após receber, a ligação do cliente, o servidor valida o utilizador que se ligar. O servidor recebe o *username* e com este gera o *token* com base nas suas credenciais existentes na base de dados RADIUS. Compara os dois *tokens* de autenticação (*Authtoken* recebido e gerado) e, se estes coincidirem, irá cifrar o ficheiro de configuração de rede sem fio segura com a chave gerada no servidor. Depois, envia o ficheiro cifrado para o cliente e espera por mais utilizadores.

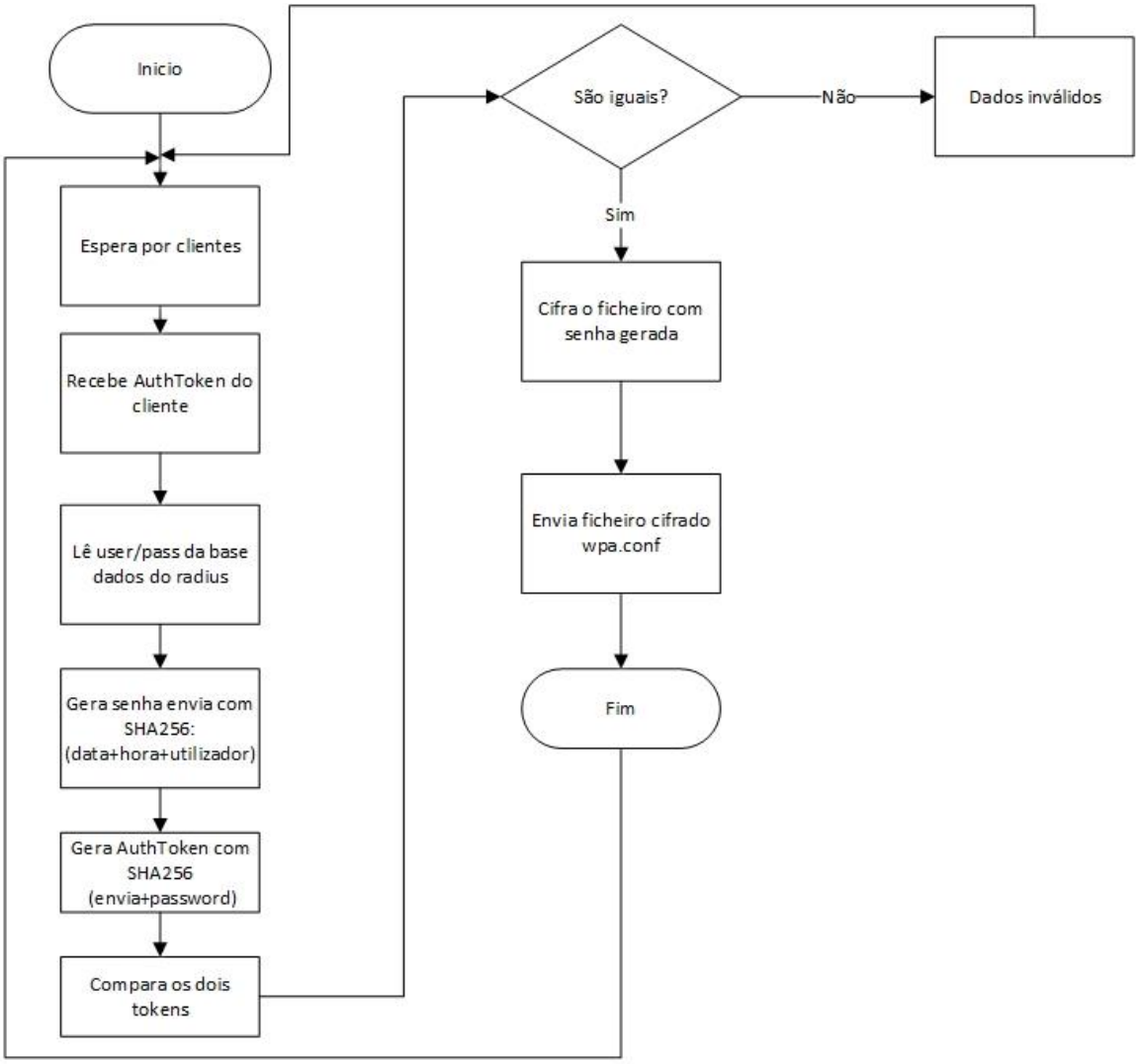


Figura 12 – Fluxograma relativo ao Servidor.

As aplicações devem estar disponíveis tanto no cliente (Client.java) como no operador de autenticação (Server.java). A aplicação do lado do servidor é instalada uma vez. A aplicação java lado do cliente é disponibilizada para os clientes com um *link* na página de *login* do *captive portal* e deve ser instalado uma vez.

A Figura 13 representa o funcionamento da proposta de solução. Em particular a Figura 3 mostra a diferença entre o funcionamento padrão do *captive portal* e a solução proposta, a partir do passo 8 (Figura 13). Os passos anteriores do funcionamento padrão do *captive portal* são necessários e assume-se a mesma ordem.

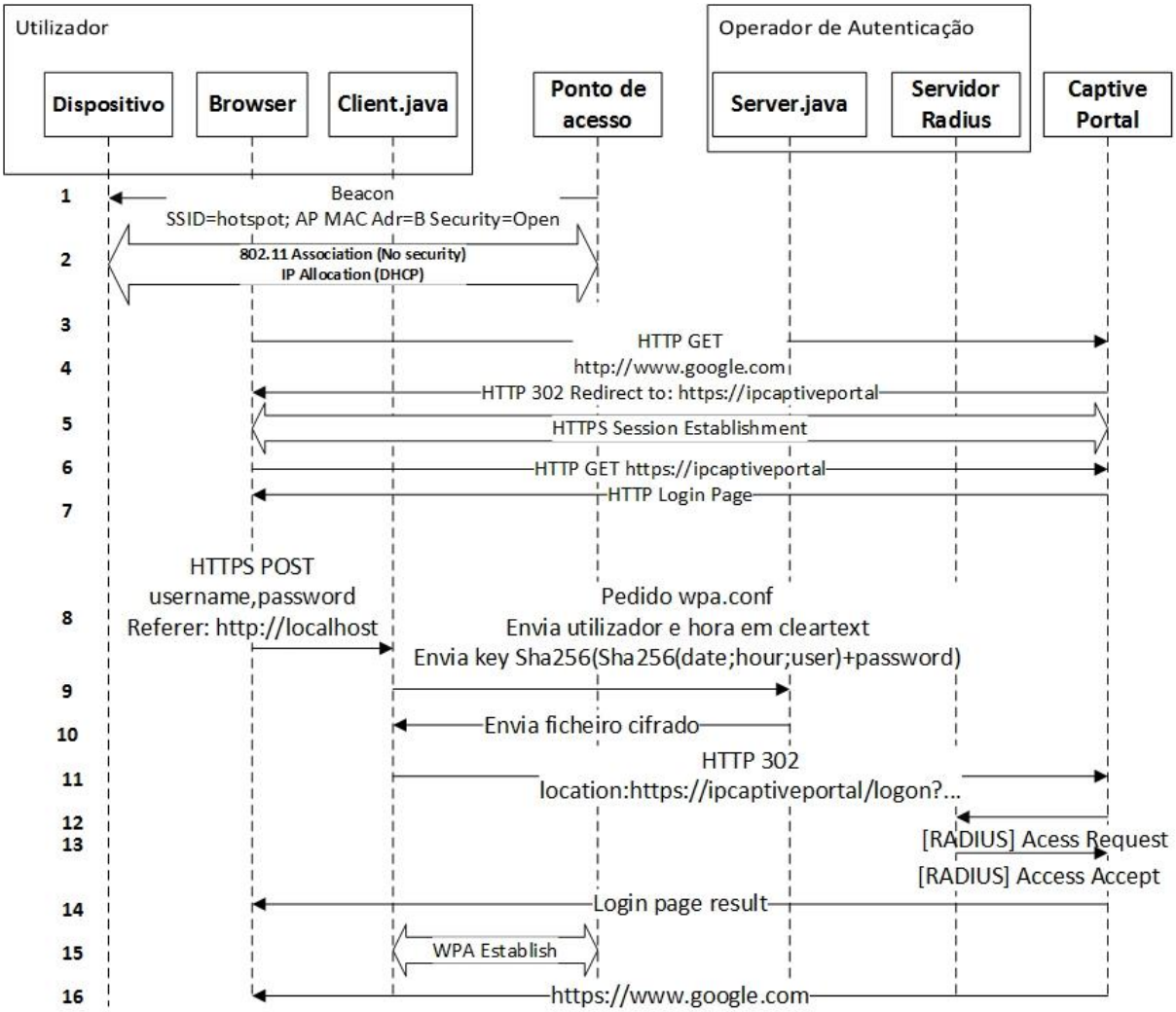


Figura 13 – Representação do funcionamento da proposta de solução.

A aplicação java lado do cliente é introduzida de modo a que o utilizador não necessite de apresentar as suas credenciais mais do que uma vez. O redireccionamento do login (passo 8 da Figura 13Figura 3) é feito para que o navegador faça a solicitação POST para a aplicação do lado do cliente java. Assim sendo, é possível que a aplicação java lado do cliente obtenha as credenciais dos utilizadores que ele vai precisar para a decifragem do ficheiro de configuração. As credenciais do utilizador são usadas pela aplicação java lado do servidor para cifrar o ficheiro de configuração antes da sua transmissão para o cliente (passo 10).

Em seguida, a aplicação java do cliente irá emitir a solicitação HTTP 302 como se fosse o navegador do cliente. Isto irá permitir o normal funcionamento contínuo do *captive portal*, como por exemplo, verificar as credenciais do utilizador e remoção de regras de redireccionamento HTTP.

Para concluir o processo, a aplicação java lado do cliente, agora na posse do ficheiro de configuração, irá acionar automaticamente a ligação à nova rede sem fio de uma forma segura. Após o servidor RADIUS validar as credenciais, o utilizador é informado que o pedido foi aceite e uma configuração de rede WPA é utilizada.

4.2. PROVA DE CONCEITO

A prova de conceito da solução proposta recorreu à instalação (ver Figura 14) num PC com 4GB de memória RAM, um processador dual-core 2.0 GHz e correndo o sistema operativo: Windows 7 (64 bits). Duas máquinas virtuais foram configuradas com o CentOS 6.6 como sistema operativo (32 bits), uma representando o sistema do cliente e outra integrando as funcionalidades, tanto do operador de *captive portal* como o operador de autenticação.

A máquina virtual do cliente foi configurada com 1506 MB de RAM virtual. A outra máquina virtual foi configurada com 1506 MB de RAM virtual, sendo instalados nela o *captive portal* Coovachili (versão 1.3.0), o FreeRADIUS servidor RADIUS (versão 2.2.0) e o pacote hostapd (versão 2.4) para simular o AP.

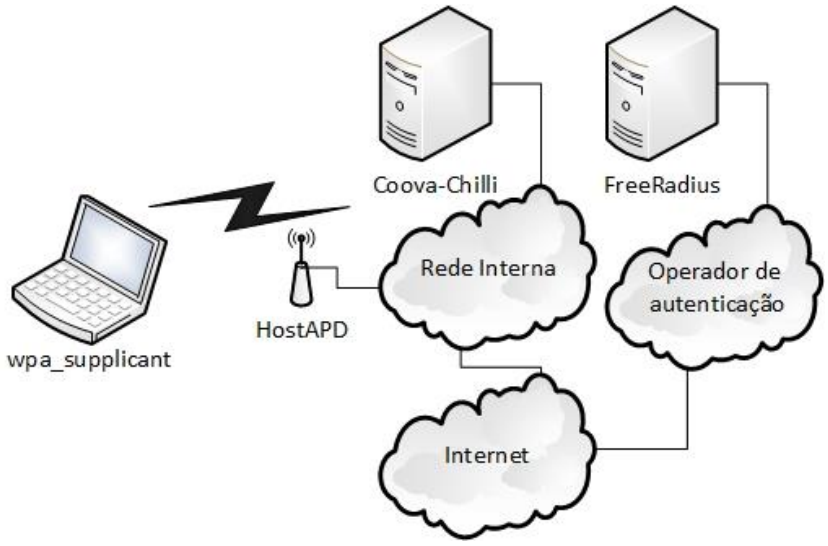


Figura 14 – Representação da instalação da prova de conceito.

Relativamente às opções do software instalado:

- O FreeRadius [43] é servidor RADIUS mais utilizado em Linux, este é uma implementação do protocolo RADIUS, com bom desempenho e várias funcionalidades. Opção recaiu neste servidor devido à sua vasta biblioteca e ferramentas de suporte relacionadas com o protocolo RADIUS.

- A escolha do *software* utilizado para *captive portal* recaiu no coovachilli [44] sendo este *open-source* e, por isso, existindo a possibilidade de acesso ao código fonte, assim como ter um vasto apoio documental no seio da comunidade responsável pelo seu desenvolvimento. É de fácil portabilidade, caso seja necessária uma migração para outra distribuição Linux, e possui uma interface amigável. Relativamente ao contexto da interface com os utilizadores clientes, tem bom suporte com páginas adaptáveis a dispositivos móveis.

- O hostapd [45] (*Host Access Point Daemon*) é um *software* desenhado para ser um *daemon*, programa que é executado em segundo plano, capaz de transformar placas de interface de rede normais em pontos de acesso e servidores de autenticação. A versão atual suporta Linux (Host AP, madwifi, drivers baseados em mac80211) e FreeBSD (net80211).

- O wpa_supplicant é um suplicante WPA para Linux, BSD e Windows com suporte para WPA e WPA2. Como suplicante é o componente IEEE 802.1x/WPA que é utilizado do lado do cliente. Implementa a negociação da chave com um Autenticador WPA, e controla o *roaming* e IEEE 802.11 autenticação/associação do *driver* de WLAN. O wpa_supplicant foi desenhado para operar como um *daemon* e age como o componente *backend* controlando a ligação sem fios.

A página de *login* do *captive portal* (Figura 15) adotada foi alterada para que o utilizador seja capaz de selecionar a solução proposta como um modo de autenticação. A operação normal é mantida e continua disponível. O utilizador deve selecionar "Usar conexão segura", para usar a solução proposta.

Figura 15 – Representação da página de *login*.

Para além da alteração da página para que seja possível utilizar a conexão segura, foi configurado um certificado SSL de modo a estabelecer-se sessão HTTPS gerando-se uma chave privada RSA de 2048 bits.

A Figura 16 mostra uma captura de rede de uma associação bem-sucedida com rede WPA. A captura de rede começa com a mensagem de início EAPOL tradicional de EAP-TTLS e termina com a autenticação do cliente sucesso.

	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	CadmusCo_37:a8:74	Nearest	EAPOL	60	Start
2	0.006411000	CadmusCo_50:d3:29	Nearest	EAP	23	Request, Identity
3	0.008295000	CadmusCo_37:a8:74	Nearest	EAP	60	Response, Identity
4	0.052028000	CadmusCo_50:d3:29	Nearest	EAP	40	Request, MD5-Challenge EAP (EAP-MD5-CHALLENGE)
5	0.054675000	CadmusCo_37:a8:74	Nearest	EAP	60	Response, Legacy Nak (Response Only)
6	0.065992000	CadmusCo_50:d3:29	Nearest	EAP	24	Request, Tunneled TLS EAP (EAP-TTLS)
7	0.069788000	CadmusCo_37:a8:74	Nearest	TLSv1	179	Client Hello
8	0.122689000	CadmusCo_50:d3:29	Nearest	TLSv1	1042	Server Hello, Certificate, Server Key Exchange, Server Hello Done
9	0.125114000	CadmusCo_37:a8:74	Nearest	EAP	60	Response, Tunneled TLS EAP (EAP-TTLS)
10	0.128726000	CadmusCo_50:d3:29	Nearest	TLSv1	1042	Server Hello, Certificate, Server Key Exchange, Server Hello Done
11	0.130067000	CadmusCo_37:a8:74	Nearest	EAP	60	Response, Tunneled TLS EAP (EAP-TTLS)
12	0.153908000	CadmusCo_50:d3:29	Nearest	TLSv1	559	Server Hello, Certificate, Server Key Exchange, Server Hello Done
13	0.177588000	CadmusCo_37:a8:74	Nearest	TLSv1	158	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
14	0.185982000	CadmusCo_50:d3:29	Nearest	TLSv1	87	Change Cipher Spec, Encrypted Handshake Message
15	0.190405000	CadmusCo_37:a8:74	Nearest	TLSv1	114	Application Data, Application Data
16	0.195083000	CadmusCo_50:d3:29	Nearest	TLSv1	97	Application Data
17	0.202250000	CadmusCo_37:a8:74	Nearest	TLSv1	130	Application Data, Application Data
18	0.207536000	CadmusCo_50:d3:29	Nearest	EAP	22	Success

Figura 16 – Representação de autenticação bem-sucedida com WPA.

5. AVALIAÇÃO EXPERIMENTAL

No cenário de teste utilizado foram realizadas alterações à instalação relativa à prova de conceito. O seguinte cenário (Figura 17), foi utilizado para obter os resultados e perceber o desempenho geral do sistema.

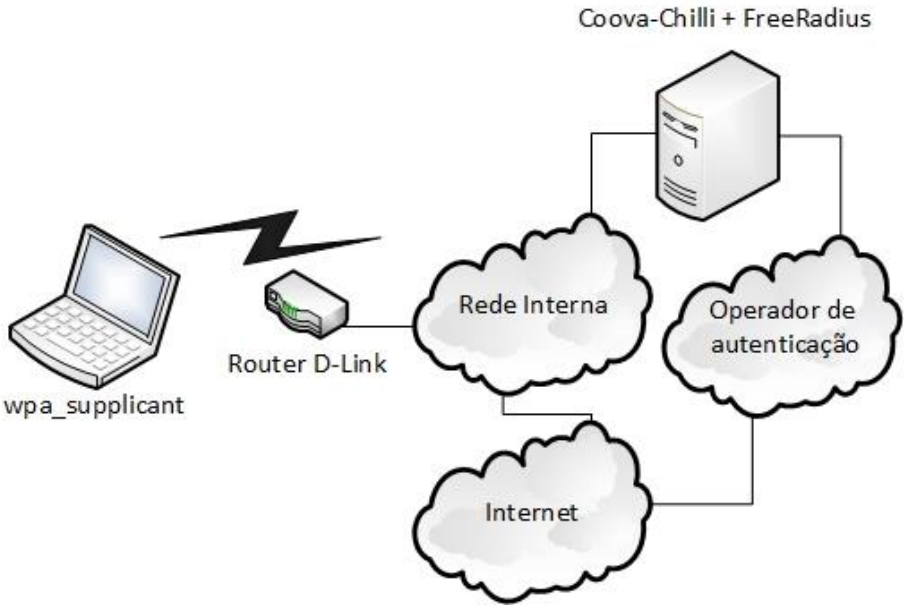


Figura 17 – Representação do cenário de testes experimentais.

Os testes foram executados em dois computadores com as mesmas características: 4 GB de RAM, um processador dual-core de 2,0 GHz, com um sistema operativo Windows 7 SP1 de 64bits, cada um com uma máquina virtual com 1506 MB de RAM e cujo sistema operativo é CentOS 6.6. O servidor e o cliente foram configurados em máquinas distintas, estando o *captive portal* e o operador de autenticação (servidor RADIUS) na primeira máquina e o cliente na segunda máquina. Para comunicação entre as máquinas foi utilizado um D-Link DSL-G604T DSL Wireless Router, acedendo-se ao *captive portal* na máquina cliente por redes sem fios. No *router* foi configurado o seu IP (dentro da gama IPs atribuída pelo *captive portal*), o seu *gateway* e foi desabilitado o DHCP para que seja a máquina servidor a atribuir os IPs pelo *captive portal* e mantendo-se a estrutura original do *CoovaChilli*.

Os testes focaram-se em determinar o tempo de execução do programa java do cliente a partir do momento em que emite a solicitação do ficheiro e até que o ficheiro esteja pronto para ser utilizado com o comando wpa_supplicant. Diferentes ensaios foram feitos utilizando diferentes versões do algoritmo AES, nomeadamente, o AES-256, o AES-192 e o AES-128

Cada teste foi executado 10 vezes e foram produzidos histogramas, para cada versão AES. O número de vezes que um valor específico foi obtido é representado como frequência absoluta.

Além disso, a frequência relativa, em proporção com o intervalo de tempo, é também evidenciado.

A Figura 18 mostra o histograma dos tempos obtidos para a versão AES-256 do algoritmo. O valor mais frequente (com 60% dos resultados obtidos) encontra-se nos intervalos 17000-19499 e 19500-21999ms.

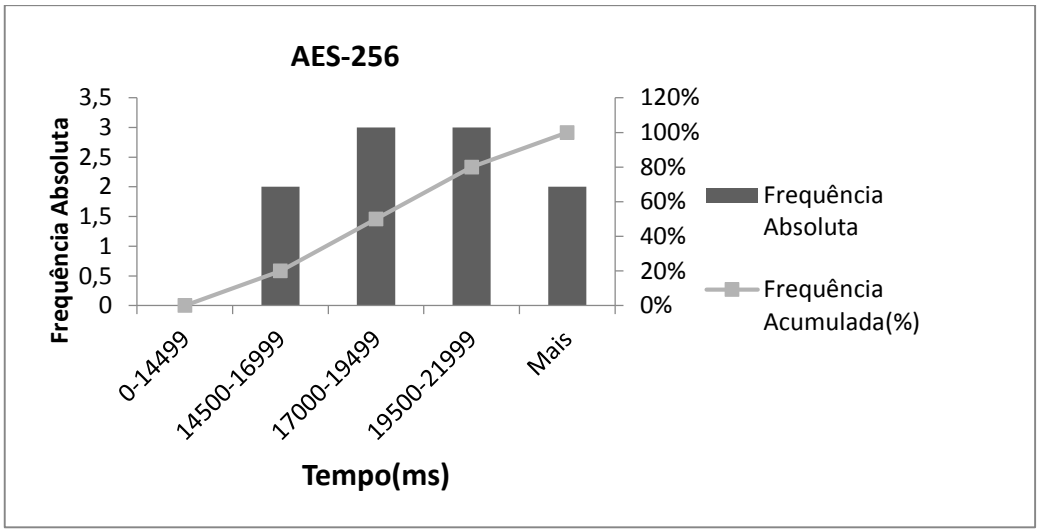


Figura 18 – Tempo de execução do programa java lado do Cliente (AES 256)

A Figura 19 mostra o histograma dos tempos obtidos ao utilizar a versão AES-192 do algoritmo. O valor mais frequente (com 60% dos resultados obtidos) encontra-se no intervalo 17000-19499ms.

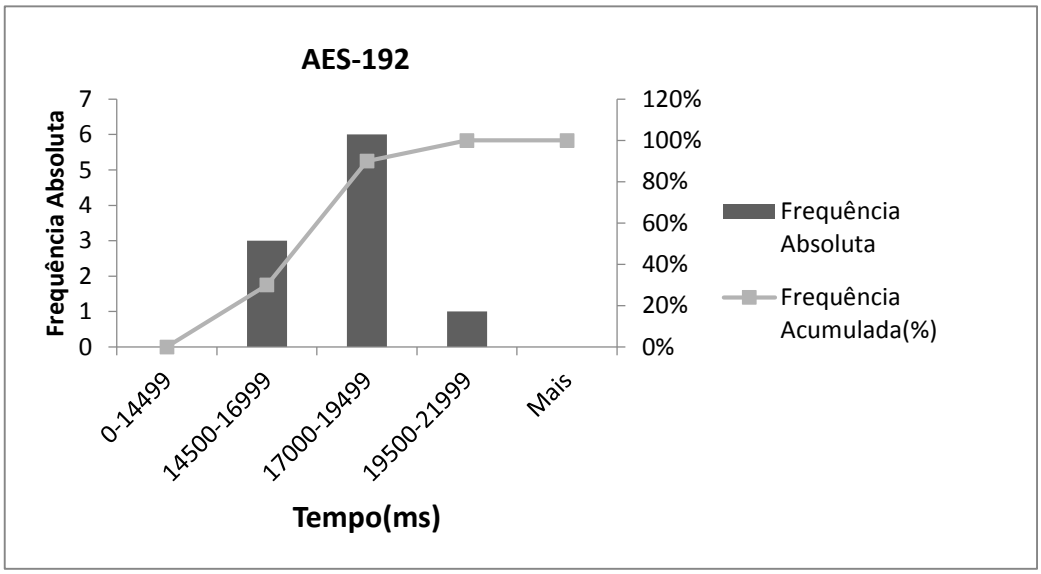


Figura 19 - Tempo de execução do programa java lado do Cliente (AES 192)

A Figura 20 mostra o histograma dos tempos obtidos ao utilizar a versão AES-128 do algoritmo. O valor mais frequente (de 90% de todos os valores) está no intervalo 14500-16999ms.

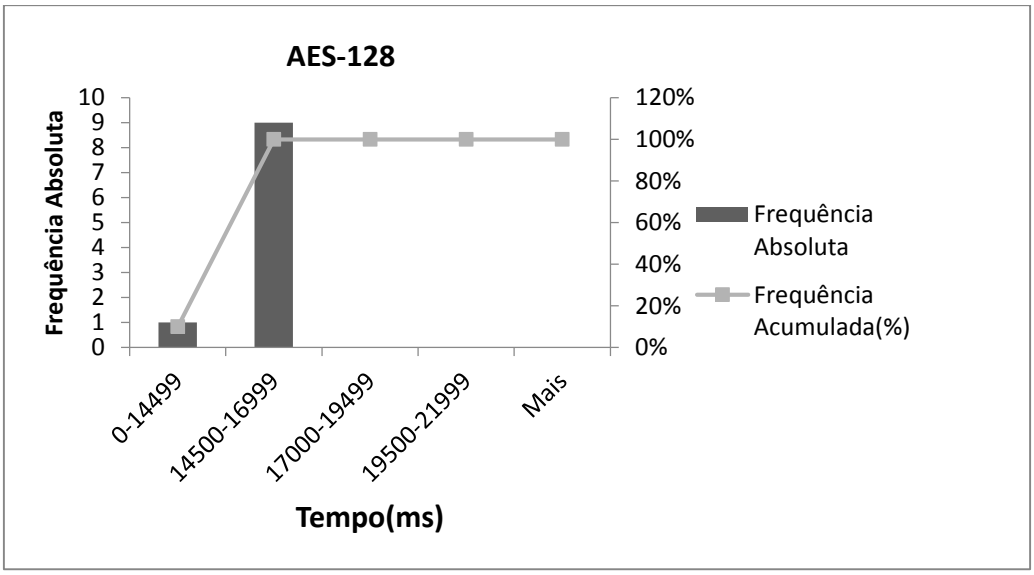


Figura 20 - Tempo de execução do programa java lado do Cliente (AES 128)

A Tabela 2 mostra a duração média por versão AES. A duração média foi de 15445 ms para a AES-128, 17956 ms para a AES-192, e para o mais exigente versão AES-256 do algoritmo.

Tabela 2 – Média do tempo de execução

Criptografia	Média (ms)
AES - 256	19968
AES - 192	17956
AES - 128	15445

De evidenciar, que apesar de mais tempo ser gasto e recursos nos algoritmos com maior tamanho de chave maior é também a segurança. Como é o caso do AES-256 quando comparado com AES-192 e AES128, e o caso do AES-192 comparando com o AES-128.

Deve-se adotar o caso mais seguro se permitir atender um número adequado de clientes.

- Testes de *stress*

Os testes de *stress* permitem avaliar o desempenho por parte do servidor Tabela 3. Foram feitos testes executando-se na máquina cliente o lançamento de 50 *threads* do programa java cliente desenvolvido, simulando-se os clientes.

Tabela 3 – Dados testes de *stress.*

Criptografia	Largura de Banda (Mbit/s)	Pedidos com sucesso (%)	Clientes/seg
AES-128	0.038	100 %	5.0
AES-192	0.090	100 %	4.5
AES-256	0.025	100 %	3.9

Relativamente aos resultados obtidos, para o AES-128, o tempo total para o atendimento dos 50 pedidos foi de 10 segundos, sendo cada cliente atendido em média em 0.2 segundos e o número de pedidos por segundo de 5. No AES-192, o tempo total para o atendimento dos 50 pedidos foi de 11 segundos, cada cliente foi atendido em média em 0.22 segundos e o número de pedidos por segundo de 4.5. No AES-256, o tempo total para o atendimento dos 50 pedidos foi de 13 segundos, cada cliente foi atendido em média em 0.26 segundos e o número de pedidos por segundo de 3.9. Os resultados obtidos em relação ao número de clientes por segundo foram bastante semelhantes, pelo que, se apresenta os resultados com uma casa decimal, para que sejam notórias as diferenças.

Quanto à largura de banda necessária, registou-se um aumento para os casos com menor segurança, isto é, menor tamanho de chave de cifragem. A largura de banda de AES-128 e AES-192 obteve valores maiores quando comparados com o obtido para AES-256. No que se refere à percentagem de pedidos com sucesso, para todos os casos de estudo da Tabela 3 foi de 100 %.

Foram ainda feitos testes com um número mais elevado de pedidos, mas inconclusivos, visto ocorrem erros de memória disponível na máquina cliente devido ao lançamento de um elevado número de pedidos e das características das máquinas de testes.

6. CONCLUSÕES

Os pontos de acesso *wireless* públicos baseados em *captive portal* são presentes em lugares como o aeroporto, estações de comboios e até mesmo no seio familiar. Os utilizadores ao ligarem-se a pontos de acessos através de ligações sem fio abertas, expõem-se a uma diminuição da sua segurança, confidencialidade e privacidade no seu uso da Internet. A adoção de redes *wireless* abertas advém da sua facilidade de uso e configuração automática.

A solução proposta mantém o uso e modo de funcionamento do *captive portal* tradicional ao permitir as configurações automáticas de redes sem fio seguras através da troca de ficheiros de configurações de rede e ajustando-os automaticamente. A solução é transparente e minimiza o seu impacto no modo como os utilizadores estão habituados a utilizar os *captive portal*.

Como prova de conceito da solução foi realizada uma implementação, e avaliada a *performance* num cenário de teste. Os resultados são apresentados e permitem concluir que a relação segurança/rapidez de execução são uma relação de compromisso. Portanto para casos, em que se pretenda maior segurança na cifragem utilizada perde-se em relação aos recursos gastos pelo sistema.

De salientar que a solução desenvolvida é, tanto quanto se sabe inovadora, e permitiu resultados bastante satisfatórios. Com os testes e experiências realizadas foi possível consumar os objetivos desta tese e demonstrar a mais-valia da solução proposta.

Relativamente ao trabalho futuro propõe-se que no protótipo final exista maior portabilidade do lado do cliente sendo possível a este estabelecer a sessão WPA em ambientes *Windows*, *Android*, *Iphone*, entre outros. Cada ambiente tem as suas especificações o que implica ficheiros de configuração distintos. Apesar da utilização do java, considerado uma linguagem multiplataforma, existe esta necessidade para que o processo possa ser o mais transparente para o utilizador. Outra possibilidade a ter em consideração numa versão futura é automatização da deteção do *captive portal* por parte dos dispositivos.

Referências Documentais

[1] Wi-Fi Alliance, “Wi-Fi Protected Access, version 2.0,” 2003.

[2] E. Sadot, L. L. Yang, and P. Zerfos, “Architecture Taxonomy for Control and Provisioning of Wireless Access Points (CAPWAP),” *IETF(Internet Eng. Task Force),* 2005.

[3] A. P. <andreipo@microsoft.com>, “Prohibiting RC4 Cipher Suites,” *IETF(Internet Eng. Task Force),* 2015.

[4] T. R. Kothaluru, “Evaluation of EAP Authentication Methods in Wired and Wireless Networks,” no. October, 2012.

[5] J. R. Vollbrecht, B. Aboba, L. J. Blunk, H. Levkowetz, and J. Carlson, “Extensible Authentication Protocol (EAP),” *IETF(Internet Eng. Task Force),* 2004.

[6] B. Aboba and D. Thaler, “What Makes For a Successful Protocol?,” *IETF(Internet Eng. Task Force),* 2008.

[7] J. Amorim, “Rede EDUROAM baseada em FreeRadius com EAP-TTLS,” 2012. [Online]. Available: http://repositorio-aberto.up.pt/bitstream/10216/65317/2/11344.pdf. [Accessed: 26-Oct-2015].

[8] S. Kelly, S. Frankel, and R. Glenn, “The AES-CBC Cipher Algorithm and Its Use with IPsec,” *IETF(Internet Eng. Task Force),* 2003.

[9] C. Boulton, S. Pietro Romano, H. Schulzrinne, and M. Barnes, “Centralized Conferencing Manipulation Protocol,” *IETF(Internet Eng. Task Force),* 2012.

[10] L. Barreto, “Segurança em arquitecturas de rede para acesso sem fios,” 2005. [Online]. Available: http://portal.ipvc.pt/images/ipvc/esce/docentes/lbarreto_1/Tese.pdf. [Accessed: 26-Oct-2015].

[11] P. Funk and S. Blake-Wilson, “Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0),” *IETF(Internet Eng. Task Force).*

[12] G. Z. <gwz@acm.org>, “Microsoft PPP CHAP Extensions, Version 2,” *IETF(Internet Eng. Task Force),* 2000.

[13] J. Almeida, “Redes Domésticas Seguras,” 2009. [Online]. Available: http://repositorio-aberto.up.pt/bitstream/10216/66741/1/000136633.pdf. [Accessed: 26-Oct-2015].

[14] S. Willens, A. C. Rubens, C. Rigney, and W. A. Simpson, “RFC [2865] - Remote Authentication Dial In User Service (RADIUS),” *IETF(Internet Eng. Task Force)*.

[15] D. Mitton, “RFC [2882]-Network Access Servers Requirements: Extended RADIUS Practices,” *IETF(Internet Eng. Task Force)*. [Online]. Available: http://ietf.org/rfc/rfc2882.txt. [Accessed: 25-Sep-2015].

[16] B. Oliveira, “Redes e serviços IP para comunidades de utilizadores,” 2008. [Online]. Available: https://repositorio-aberto.up.pt/bitstream/10216/60244/2/Texto integral.pdf. [Accessed: 26-Oct-2015].

[17] D. Lloyd, B., L&A, Simpson, W., “PPP Authentication Protocols,” *IETF(Internet Eng. Task Force)*, 1992. [Online]. Available: https://www.ietf.org/rfc/rfc1334.txt. [Accessed: 26-Sep-2015].

[18] G. Zorn and S. Cobb, “Microsoft PPP CHAP Extensions,” *IETF(Internet Eng. Task Force)*, 1998.

[19] N. Antunes, “Frontend Web 2.0 para Gestão de RADIUS,” 2009. [Online]. Available: http://paginas.fe.up.pt/~ee04199/Frontend Web 2.0 para Gestao de RADIUS.pdf. [Accessed: 26-Oct-2015].

[20] A. Asheralieva, T. J. Erke, and K. Kilkki, “Traffic characterization and service performance in FON network,” in *2009 1st International Conference on Future Information Networks, ICFIN 2009*, 2009, pp. 285–291.

[21] G. Camponovo and A. Picco-Schwendener, “Motivations of hybrid wireless community participants: A qualitative analysis of Swiss FON members,” *Proc. - 2011 10th Int. Conf. Mob. Business, ICMB 2011*, pp. 253–262, 2011.

[22] L. Mamakos, D. Simone, R. Wheeler, D. Carrel, J. Evarts, and K. Lidl, “A Method for Transmitting PPP Over Ethernet (PPPoE),” *IETF(Internet Eng. Task Force)*.

[23] “Em que diferem os concentradores, comutadores, routers e pontos de acesso? - Ajuda do Windows.” [Online]. Available: http://windows.microsoft.com/pt-pt/windows/hubs-switches-routers-access-points-differ#1TC=windows-7. [Accessed: 27-Oct-2015].

[24] “IEEE 802.1X Open Source Implementation,” 2010. [Online]. Available: http://open1x.sourceforge.net/.

[25] “Linux WPA Supplicant (IEEE 802.1X, WPA, WPA2, RSN, IEEE 802.11i).” [Online]. Available: http://w1.fi/wpa_supplicant/. [Accessed: 25-Sep-2015].

[26] M. Barbosa, “Criptografia Módulo I – Terminologia,” 2006. [Online]. Available: http://twiki.di.uminho.pt/twiki/pub/Education/Criptografia/CriptografiaBiomedica0607/Cripto-Mod1.pdf. [Accessed: 26-Oct-2015].

[27] L. Matos, “Criptologia – Segurança na Internet,” 2004. [Online]. Available: http://www.dei.isep.ipp.pt/~paf/proj/Set2004/Criptologia Seguranca.pdf. [Accessed: 26-Oct-2015].

[28] A. Lhamas, “Conceitos de Segurança.”

[29] J. Alves, P. Campos, and P. Brito, *O futuro da Internet: estado da arte e tendências de evolução*. Centro Atlantico, 1999.

[30] M. Barbosa and J. Almeida, “Criptografia Aplicada Criptografia e Segurança da Informação,” 2007. [Online]. Available: http://wiki.di.uminho.pt/twiki/pub/Education/Criptografia/CriptografiaAplicada0708/mod1.pdf. [Accessed: 26-Oct-2015].

[31] N. Pereira, “Administração de Sistemas Informáticos 1 - Aula 12,” 2005.

[32] M. Fernandes, “Extensões para Design de Hardware Digital em Aplicações Aeroespaciais,” 2012. [Online]. Available: https://repositorium.sdum.uminho.pt/bitstream/1822/25727/1/Disserta%C3%A7%C3%A3o_Marta Patr%C3%ADcia Teixeira Fernandes.pdf. [Accessed: 26-Oct-2015].

[33] A. Moreira, “Criptografia,” 2002. [Online]. Available: http://www.dei.isep.ipp.pt/~andre/documentos/criptografia.html. [Accessed: 26-Oct-2015].

[34] R. Rivest, “The RC5 Encryption Algorithm.” [Online]. Available: https://people.csail.mit.edu/rivest/Rivest-rc5rev.pdf. [Accessed: 26-Oct-2015].

[35] B. Pokhali, “Satellite On-board Encryption,” 2007.

[36] R. Coelho, “Segurança em Redes Conceitos de criptografia 2 – Cifra assimétrica,” 2011. [Online]. Available: http://pwp.net.ipl.pt/sc/rcoelho/EI/SRC-2014/03.2 - Conceitos de criptografia - Cifras assim%C3%A9tricas.pdf. [Accessed: 26-Oct-2015].

[37] “RSA Laboratories - TWIRL and RSA Key Size.” [Online]. Available: http://www.emc.com/emc-plus/rsa-labs/historical/twirl-and-rsa-key-size.htm. [Accessed: 26-Oct-2015].

[38] K. Maletsky, “RSA vs ECC Comparison for Embedded Systems.”

[39] J. Dias, “Segurança em Redes WiFi.” [Online]. Available: https://web.fe.up.pt/~mricardo/07_08/cmov/slides/security-wireless-networks.pdf. [Accessed: 26-Oct-2015].

[40] N. Lopes, “Avaliação da segurança e desempenho criptográfico de sistemas baseados em DNSSEC,” 2006. [Online]. Available: http://repositorio-aberto.up.pt/bitstream/10216/60741/1/000135648.pdf. [Accessed: 26-Oct-2015].

[41] “SCEE - Sistema de Certificação Electrónica do Estado.” [Online]. Available: http://www.scee.gov.pt/rep/defini%C3%A7%C3%B5es/. [Accessed: 24-Oct-2015].

[42] “Criptografia Assimétrica.” [Online]. Available: http://wiki.di.uminho.pt/twiki/pub/Education/Criptografia/CriptografiaAplicada0708/CSI-PubKey.pdf. [Accessed: 26-Oct-2015].

[43] “FreeRADIUS: The world’s most popular RADIUS Server.” [Online]. Available: http://freeradius.org/. [Accessed: 25-Sep-2015].

[44] “CoovaChilli, an open source captive portal access controller.” [Online]. Available: https://coova.github.io/CoovaChilli/. [Accessed: 25-Sep-2015].

[45] “hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator.” [Online]. Available: https://w1.fi/hostapd/. [Accessed: 25-Sep-2015].

Anexo A. Cliente java

```
package client;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.MessageDigest;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Properties;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
```

```
public class Cliente {

    private String reportDate, enviarString;
    private final static int SOCKET_PORT = 4444;
    private final static String SERVER = "10.1.0.1";
    private final static String FILE_TO_RECEIVED =
"/home/andre/Documents/wpa.conf";
    private final static int FILE_SIZE = 6022386;
    private Socket sock = null;
    private FileOutputStream fos = null;
    private BufferedOutputStream bos = null;
    private String[] extraUrl;

    public byte[] generateHASH(byte[] message) throws
Exception {
        MessageDigest messageDigest =
MessageDigest.getInstance("SHA-256");
        byte[] hash = messageDigest.digest(message);
        //byte[] hash2 = Arrays.copyOf(hash, 32); // 256
key
        // byte[] hash2 = Arrays.copyOf(hash, 16); // 128
key
         byte[] hash2 = Arrays.copyOf(hash, 24); // 192
key
        return hash2;
    }
```

```
    public byte[] decrypt(byte[] encMsgtoDec, byte[] key)
throws Exception {

        SecretKeySpec       secretKeySpec       =       new
SecretKeySpec(generateHASH(key), "AES");
        byte firstiv[] = generateHASH(key);
        byte iv[] = new byte[16];
          System.arraycopy(firstiv, 0, iv, 0, 16);
        String cipherALG = "AES/CBC/PKCS5padding";
        Cipher cipher = Cipher.getInstance(cipherALG);
        String string = cipher.getAlgorithm();

        if (string.contains("CBC")) {
            IvParameterSpec    ivParameterSpec    =    new
IvParameterSpec(iv);
            cipher.init(Cipher.DECRYPT_MODE,
secretKeySpec, ivParameterSpec);
        } else {
            cipher.init(Cipher.DECRYPT_MODE,
secretKeySpec);
        }
        byte[] decMsg = cipher.doFinal(encMsgtoDec);
        return decMsg;
    }
    public void Capture() throws IOException {

        int port = 80;
        ServerSocket       serverSocket       =       new
ServerSocket(port);
        Socket clientSocket = serverSocket.accept();
      BufferedReader    in    =    new    BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        BufferedWriter   out   =   new   BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream()));
         String content;
        Properties headers = new Properties();
```

```
        String inputLine = null;

        while ((inputLine = in.readLine()) != null) {
            String[] headerVar = inputLine.split(":");
            if (headerVar.length > 1) {
                headers.setProperty(headerVar[0].trim(),
headerVar[1].trim());
            }
            inputTotal.append(inputLine);
            if (headers.getProperty("Content-Length") !=
null) {
                content = headers.getProperty("Content-
Length");
                int len = Integer.parseInt(content);
                char[] characters = new char[(len + 20)];
                int bytesread = in.read(characters);
                String n;
                n = Arrays.toString(characters);
                String formatedString = n.toString()
                        .replace(",", "")
                        .replace("[", "")
                        .replace("]", "")
                        .replace(" ", "")
                        .trim();
                extraUrl = formatedString.split("&");
                String s2 =
"https://10.1.0.1/coova_json/login.php";
                out.write("<html>");
                out.write("<body
onload='document.forms[\"login\"].submit()'>");
                out.write("<form name='login' action='" +
s2 + "' method='post'>");
```

```
                out.write("<input type=\"hidden\"
name=\"username\" value=" + (extraUrl[2].split("=")[1]) +
" />");
                out.write("<input type=\"hidden\"
name=\"password\" value=" + (extraUrl[3].split("=")[1]) +
" />");
                out.write("<input type=\"hidden\"
name=\"challenge\" value=" + (extraUrl[0].split("=")[1]) +
" />");
                out.write("<input type=\"hidden\"
name=\"userurl\" value=" + (extraUrl[1].split("=")[1]) + "
/>");
                out.write("</form>");
                out.write("</body>");
                out.write("</html>");
                break;
            }
        }
        System.err.println("A redirecionar autenticação");
        out.close();
        in.close();
        clientSocket.close();
    }
    public void Date() {
        DateFormat df = new SimpleDateFormat("dd MMM yyyy
HH:mm");
        java.util.Date today =
Calendar.getInstance().getTime();
        reportDate = df.format(today);
    }
```

```
    public void Format() throws Exception {

        String user = (extraUrl[2].split("=")[1]);
        String pass = (extraUrl[3].split("=")[1]);
        String first = user + reportDate;
        byte[] firstHash = generateHASH(first.getBytes());
        String firstStringHash = new String(firstHash,
"UTF-8");
        String secondStringHash = firstStringHash + pass;
        byte[] secondHash =
generateHASH(secondStringHash.getBytes());
        StringBuilder sb = new StringBuilder();
        for (byte b : secondHash) {
            sb.append(String.format("%02X", b & 0xff));
        }
        enviarString = sb.toString();
    }
    public void receive() throws IOException {
        int bytesRead;
        int current = 0;
        String chave = enviarString;

        try {

            System.out.println("Connecting...");
            byte[] mybytearray = new byte[FILE_SIZE];
            InputStream is = sock.getInputStream();
            bytesRead = is.read(mybytearray, 0,
mybytearray.length);
            current = bytesRead;
            String err = new String(mybytearray);
```

```
if (err.contains("invalido")) {
                System.out.println("Utilizador não
valido.");
                return;
            }
            fos = new FileOutputStream(FILE_TO_RECEIVED);
            bos = new BufferedOutputStream(fos);
            bos.write(mybytearray, 0, current);
            bos.flush();
            File myFile = new File(FILE_TO_RECEIVED);
            byte[] myEncByteArray = new byte[(int)
myFile.length()];
            FileInputStream fis = new
FileInputStream(myFile);
            BufferedInputStream bis = new
BufferedInputStream(fis);
            bis.read(myEncByteArray, 0,
myEncByteArray.length);
            byte[] decr = decrypt(myEncByteArray,
chave.getBytes());
            FileOutputStream decryptedData = new
FileOutputStream(FILE_TO_RECEIVED);
            decryptedData.write(decr);
            decryptedData.close();
            bos.flush();
            System.out.println("File " + FILE_TO_RECEIVED
+ " downloaded ("
                    + current + " bytes read)");

         InitWPA();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
        }
```

```
    public void sendsha() throws IOException {

        String str = "A iniciar";
        sock = new Socket(SERVER, SOCKET_PORT);
        BufferedReader br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
        PrintWriter pw = new
PrintWriter(sock.getOutputStream(), true);
        pw.println(str);
        try {
            while ((str = br.readLine()) != null) {
                String Ucleartext =
(extraUrl[2].split("=")[1]);
                String Dcleartext = reportDate;
                String verify = enviarString;
                pw.println(Ucleartext);
                pw.println(Dcleartext);
                pw.println(verify);
                pw.println("bye");

                if (str.equals("bye")) {
                    break;
                }
            }
        } catch (IOException e1) {
            System.out.println("Erro no envio das
strings");
            e1.printStackTrace();
        }
    }
```

```
   public void InitWPA() {
                try {
                Runtime rt = Runtime.getRuntime();
                Process pr = rt.exec("sudo wpa_supplicant
-c /home/andre/Documents/wpa.conf -D wired -i eth2");

                BufferedReader inputcommand = new
BufferedReader(new
InputStreamReader(pr.getInputStream()));
                String command=null;
                while((command=inputcommand.readLine()) !=
null) {
                    System.out.println(command);
                }

                int exitVal = pr.waitFor();
                System.out.println("Exited with error code
"+exitVal);
            } catch(Exception e) {
                System.out.println(e.toString());
                e.printStackTrace();
            }
    }

    public static void main(String[] args) throws
IOException, Exception {
        Cliente c = new Cliente();
        c.Capture();
        c.Date();
        c.Format();
        c.sendsha();
        c.receive();
    }
}
```

Anexo B. Servidor java

```
package server;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {
   private static ServerSocket servSocket = null;
   private static int SOCKET_PORT=4444;

    public static void main(String[] args) throws
IOException {
        servSocket = new ServerSocket(SOCKET_PORT);
        try {
            while (true) {
                System.out.println("À espera de uma
conexão em " + SOCKET_PORT + " . . .");
                Socket fromClient = servSocket.accept();
                System.out.println("Ligação aceite");
                ServerThread serverThread = new
ServerThread(fromClient);
                Thread thread1 = new Thread(serverThread);
                thread1.start();
            }
        } catch (Exception e) {
            System.out.println("Thread interrupted.");
        }
    }
}
```

Anexo C. Servidor Thread java

```
package server;


import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.MessageDigest;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Formatter;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
```

```
public class ServerThread implements Runnable {

    private String reportDate, StringGerada;
    private byte[] HashGerado, hashRecebido, serverhash;
    private int line;
    private ArrayList<String> list = new
ArrayList<String>();
    private ArrayList<ArrayList<String>> twoDimList = new
ArrayList<ArrayList<String>>();
    private ArrayList<String> users = new
ArrayList<String>();
    private ArrayList<String> passwords = new
ArrayList<String>();
    private String userRecebido, dataRecebida,
chaveRecebida;
    private final Socket fromClientSocket;
    private PrintWriter pw;
    private  BufferedReader br;
    private FileInputStream fis = null;
    private BufferedInputStream bis = null;
    private OutputStream os = null;

    public static byte[] generateHASH(byte[] message)
throws Exception {
        MessageDigest messageDigest =
MessageDigest.getInstance("SHA-256");
        byte[] hash = messageDigest.digest(message);
      //  byte[] hash2 = Arrays.copyOf(hash, 32); // 256
key
        byte[] hash2 = Arrays.copyOf(hash, 16); // 128 key
      //  byte[] hash2 = Arrays.copyOf(hash, 24); // 192
key
        return hash2;
    }
```

```
    public static byte[] encrypt(byte[] msg, byte[] key)
throws Exception {
        SecretKeySpec secretKeySpec = new
SecretKeySpec(generateHASH(key), "AES");
        byte firstiv[] = generateHASH(key);
        byte iv[] = new byte[16];
        System.arraycopy(firstiv, 0, iv, 0, 16);
        String cipherALG = "AES/CBC/PKCS5padding";
        Cipher cipher = Cipher.getInstance(cipherALG);
        String string = cipher.getAlgorithm();
        if (string.contains("CBC")) {

            IvParameterSpec ivParameterSpec = new
IvParameterSpec(iv);
            cipher.init(Cipher.ENCRYPT_MODE,
secretKeySpec, ivParameterSpec);
        } else {
            cipher.init(Cipher.ENCRYPT_MODE,
secretKeySpec);
        }

        byte[] encMessage = cipher.doFinal(msg);
        return encMessage;
    }
```

```
    public void CompareUsers() throws Exception {
        for (line = 0; line < users.size(); line++) {
            if
(userRecebido.equals(twoDimList.get(0).get(line))) {
                System.out.println("*** O utilizador n°" +
line + " tem esse nome ***");
                FormatSHA();
                hashRecebido = chaveRecebida.getBytes();
                HashGerado = StringGerada.getBytes();
                if (MessageDigest.isEqual(hashRecebido,
hashRecebido)) {
                    System.out.println("*** O utilizador
n°" + line + " tem essa chave ***");
                    SendFile();
                    break;
                } else {
                    System.out.println("Password
invalida");
                }
            } else {
                if (line == (users.size() - 1)) {
                    System.out.println("Utilizador
inválido");
                    enviaUtilizadorInvalido();
                }
            }
        }
    }
```

```
    public void enviaUtilizadorInvalido(){
        String str=null;
        BufferedReader br;
        try {
            br = new BufferedReader(new
InputStreamReader(fromClientSocket.getInputStream()));
            PrintWriter pw = new
PrintWriter(fromClientSocket.getOutputStream(), true);
        pw.println("invalido");
        if (fromClientSocket != null) {
                    fromClientSocket.close();
                }
        } catch (IOException ex) {
Logger.getLogger(ServerThread.class.getName()).log(Level.S
EVERE, null, ex);
        }
    }
    public void GenerateDate() {
        DateFormat df = new SimpleDateFormat("dd MMM yyyy
HH:mm");
        java.util.Date today =
Calendar.getInstance().getTime();
        reportDate = df.format(today);
    }
```

```
    public void FormatSHA() throws Exception {

        String user = users.get(line);
        String pass = passwords.get(line);
        String first = user + reportDate;
        byte[] firstHash = generateHASH(first.getBytes());
        String firstStringHash = new String(firstHash,
"UTF-8");
        String secondStringHash = firstStringHash + pass;
        byte[] secondHash =
generateHASH(secondStringHash.getBytes());
        StringBuilder sb = new StringBuilder();
        for (byte b : secondHash) {
            sb.append(String.format("%02X", b & 0xff));
        }
        StringGerada = sb.toString();
    }
    public void ReadUsers() throws Exception {
        String str;

        try {
            BufferedReader bufferedReader = new
BufferedReader(new FileReader("/etc/raddb/users"));
            while ((str = bufferedReader.readLine()) !=
null) {
                String linha = str.replace("\"", "");
                list.add(linha);
            }
```

```
            for (line = 0; line < list.size(); line++) {
                String[] parts = list.get(line).split("
");
                if (parts.length == 4) {
                    users.add(parts[0]);
                    passwords.add(parts[3]);
                } else {
                    break;
                }
            }
            twoDimList.add(new ArrayList<>(users));
            twoDimList.add(new ArrayList<>(passwords));
            CompareUsers();
            bufferedReader.close();

        } catch (FileNotFoundException ex) {
            System.out.println("Unable to open file");
        } catch (IOException ex) {
            System.out.println("Error reading file");
        }
    }
```

```
    public void ReceiveSHA() throws IOException {

        String str;
        pw = new
PrintWriter(fromClientSocket.getOutputStream(), true);
        br = new BufferedReader(new
InputStreamReader(fromClientSocket.getInputStream()));

        ArrayList<String> CleartextR = new
ArrayList<String>();

        try {
            while ((str = br.readLine()) != null) {

                CleartextR.add(str);
                if (str.equals("bye")) {
                    pw.println("bye");
                    break;
                } else {
                    str = "Server devolve " + str;
                    pw.println(str);
                }
            }
        }
       catch(IOException ex) {
           System.out.println("Erro a Enviar as string");
       }

            userRecebido = CleartextR.get(1);
            dataRecebida = CleartextR.get(2);
            chaveRecebida = CleartextR.get(3);
    }
```

```
   public void SendFile() throws IOException {
        String chave = StringGerada;
        try {
            System.out.println("Waiting...");
            try {
                System.out.println("Accepted connection
for SendFile : " + fromClientSocket);
                byte[] mybytearray = createfile();
                byte[] encFile = encrypt(mybytearray,
chave.getBytes());
                os = fromClientSocket.getOutputStream();
                System.out.println("Sending generated
file" + "(" + encFile.length + " bytes)");
                os.write(encFile, 0, encFile.length);
                os.flush();
                System.out.println("Done.");
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                if (bis != null) {
                    bis.close();
                }
                if (os != null) {
                    os.close();
                }
                if (fromClientSocket != null) {
                    fromClientSocket.close();
                }
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {}
}
```

```
    public byte[] createfile(){
        String user = twoDimList.get(0).get(line);
        String pass = twoDimList.get(1).get(line);
        String finString = "";
        finString +=
"ctrl_interface=/var/run/wpa_supplicant"+"\n";
        finString += "ctrl_interface_group=0"+"\n";
        finString += "eapol_version=2"+"\n";
        finString += "ap_scan=0"+"\n";
        finString += "network={"+"\n";
        finString += "key_mgmt=WPA-EAP"+"\n";
        finString += "eap=TTLS"+"\n";
        finString += "identity=\""+user+"\"\n";
        finString += "password=\""+pass +"\"\n";
        finString += "priority=1"+"\n";
        finString += "eapol_flags=0"+"\n";
        finString += "}"+"\n";

        return finString.getBytes();
    }
    public ServerThread(Socket fromClient) {
        fromClientSocket = fromClient;
    }

    public void run() {
        GenerateDate();
        try {
            ReceiveSHA();
            ReadUsers();
        } catch (Exception ex) {
Logger.getLogger(ServerThread.class.getName()).log(Level.S
EVERE, null, ex);
        }
    }
}
```