

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Atualização automática de *hardware* via *over-the-air programming*

Sérgio Augusto Pires Domingues

DISSERTAÇÃO



Mestrado Integrado em Engenharia Informática e Computação

Orientador: Prof. António Miguel Pimenta Monteiro

Supervisão na AddVolt: Eng. Ricardo Soares, Eng. Leonel Araujo

22 de Julho de 2018

Atualização automática de *hardware* via *over-the-air* programming

Sérgio Augusto Pires Domingues

Mestrado Integrado em Engenharia Informática e Computação

Presidente: Nome do presidente do júri

Arguente: Altino Sampaio

Vogal: Nome do vogal do júri

22 de Julho de 2018

Abstract

Nowadays technology (namely on automotive systems and electric vehicles) is evolving at a fast and growing pace, mainly because of the public's constant demand for new features and also because of the existing market competition. Therefore, arises the urge for the companies to match their clients' needs, in a continuous and customized way through regular product updates.

Traditionally updates are done through physical access in the devices. However the current trend is evolving into an automatic remote update distribution using over-the-air (OTA) programming. This change carries advantages concerning scalability and overall efficiency of the update delivery process. Nonetheless, this advance brings challenges regarding security, data integrity preservation and communication reliability with geographically distributed devices, whose location changes in real time.

Several solutions adopt an OTA paradigm, a vast majority of its usage is present in sectors such as mobile devices industry and automotive industry. Specifically, in the automotive industry, there is a rising tendency in adopting a remote approach on the firmware updates targeting the Engine Control Unit (ECU). Also, with the growth of the Internet of Things ecosystem, emerge over-the-air applications focused on Wireless Sensor Networks.

With this thesis it is intended to demonstrate the implementation process and results of the platform prototyping, that allows AddVolt to automate the remote firmware updates distribution management, thus discarding the need of physical intervention in the devices. The implemented solution follows a client-server architecture, with its core residing in the back-end, responsible for running the data transfer protocol, which ensures security and integrity throughout the information exchange, with the help of an appropriate module, up and running on the client side.

Resumo

Hoje em dia a evolução tecnológica dos dispositivos segue um ritmo acelerado e crescente, principalmente motivado pela exigência constante de novas funcionalidades por parte do público em geral e pela concorrência existente no mercado. Neste sentido surge a necessidade, por parte das empresas, de ir ao encontro das necessidades dos seus clientes de uma forma personalizada e contínua por intermédio do lançamento regular de atualizações aos seus produtos.

Tradicionalmente as atualizações são realizadas acedendo fisicamente aos dispositivos, no entanto a tendência está a evoluir no sentido de distribuir atualizações remotamente e de forma automática através de programação *over-the-air* (OTA). Esta mudança traz vantagens em termos de eficiência e escalabilidade do processo de atualização, contudo acarreta desafios ao nível da segurança, manutenção da integridade dos produtos e fiabilidade da comunicação com dispositivos dispersos geograficamente, cuja localização varia em tempo real.

Existem várias soluções associadas a *over the air programming*, maioritariamente aplicadas nos sectores da indústria de dispositivos móveis e na indústria automóvel. Na indústria automóvel em particular, existe atualmente uma tendência crescente na prática de atualizações remotas ao nível do *firmware* da *Engine Control Unit* (ECU). O crescimento do ecossistema que compõe a *Internet of Things* (IoT), faz com que exista também uma vertente de *over the air programming* em crescimento, focada em *Wireless Sensor Networks* (WSNs).

Com esta dissertação pretende-se demonstrar o processo e resultados da prototipagem de uma plataforma, que possibilite à AddVolt automatizar a gestão de atualizações de *firmware*, suprimindo a necessidade de intervenção física nos dispositivos. A solução desenvolvida segue uma arquitetura cliente-servidor, com foco no *back-end* que é responsável pela execução de um protocolo de transferência de dados que garante a segurança e integridade na comunicação da informação, com o auxílio de um módulo apropriado, existente e em execução no cliente.

Agradecimentos

A realização desta dissertação de mestrado contou com importantes apoios de várias pessoas.

Ao Professor Miguel Monteiro, por me ter guiado ao longo deste desafio.

A toda a equipa da Addvolt, em particular ao Eng. Leonel Araújo e Eng. Ricardo Soares pelo acompanhamento mais próximo no decorrer da evolução da dissertação.

Aos meus companheiros de curso, com os quais durante o curso tive o privilégio de viver momentos incríveis!

Aos meus pais e irmã por serem a minha força motriz ao longo desta caminhada.

Sérgio Domingues

*“The real danger is not that computers will begin to think like men,
but that men will begin to think like computers.”*

Sydney J. Harris

Conteúdo

1	Introdução	1
1.1	Contexto	1
1.2	Motivação e Objetivos	2
1.3	Metodologia	2
2	Revisão Bibliográfica	3
2.1	Atualizações OTA	3
2.1.1	Atualizações OTA em dispositivos móveis	4
2.1.2	Atualizações OTA em WSNs	5
2.1.3	Atualizações OTA na indústria automóvel	6
2.2	Criptografia	8
2.2.1	<i>Hashing</i> versus encriptação	9
2.2.2	Encriptação simétrica	13
2.2.3	Encriptação assimétrica e assinaturas digitais	14
2.2.4	Métodos de transferência de chaves	17
2.2.4.1	Diffie-Hellman Key Exchange (DHKE)	17
2.2.4.2	Elliptic Curve Diffie-Hellman Key Exchange (ECDHKE)	19
2.3	Conclusão	21
3	Proposta de solução	23
3.1	Alternativas estudadas	23
3.2	Especificação da solução	24
3.3	Arquitetura	26
3.3.1	Pressupostos	26
3.3.2	Arquitetura física	27
3.3.3	Casos de uso	28
3.3.4	Dispositivo e <i>Firmware</i>	29
4	Implementação	33
4.1	Restrição do âmbito	33
4.2	Tecnologias	34
4.3	Estrutura do projeto	35
4.4	Servidor(es)	36
4.5	Protocolo	37
4.5.1	Estrutura das tramas	37
4.5.2	Segurança e criptografia	39
4.5.3	Dinâmica do protocolo	40
4.6	Interface	41

CONTEÚDO

4.7	Limitações	43
5	Resultados	45
5.1	Operação e execução	45
5.2	Interface <i>web</i>	46
5.2.1	Página de conexões	46
5.2.2	Página de carregamento de ficheiros	47
5.2.3	Páginas auxiliares	47
5.3	Métricas avaliadas	47
5.4	Testes	48
5.4.1	Demonstração base do protocolo FOTA	49
5.4.2	Validação de requisitos	49
5.4.3	Testes de escala	51
5.5	Discussão dos resultados	52
6	Conclusões	59
6.1	Conclusões	59
6.2	Lições aprendidas	60
6.3	Evolução futura	60
	Referências	63

Lista de Figuras

2.1	Diagrama de sequência base do <i>download</i> de um pacote de atualização (Open Mobile Alliance Ltd., 2007), ...	5
2.2	Representação alto nível do processo de atualização OTA de um sensor (Libelium, 2017).	6
2.3	Diagrama do custo compartilhado do <i>Engine Control Unit</i> (ECU) em relação ao custo total do automóvel (Statista, 2017).	7
2.4	Esquema de integração de produtos da Infineon num sistema automóvel dedicados a operações de SOTA (Infineon, 2017).	8
2.5	Segurança na intersecção da confidencialidade, integridade e disponibilidade. . .	9
2.6	Requisitos para uma função de <i>hashing</i> criptográfica (Stallings, 2013).	11
2.7	Propriedades de resistência de uma função de <i>hashing</i> , necessárias em mecanismos de manutenção de integridade de dados (Stallings, 2013).	11
2.8	Ataques em funções <i>hashing</i> padrão (Mironov, 2005).	12
2.9	Encriptação simétrica (Paar and Pelzl, 2010).	13
2.10	Encriptação assimétrica. Processo (a) encriptação com chave pública, (b) encriptação com chave privada (Stallings, 2013).	16
2.11	Aplicações de encriptação assimétrica (Stallings, 2013).	16
2.12	Diffie-Hellman Key Exchange (Stallings, 2013).	18
2.13	Elliptic Curve Diffie-Hellman Key Exchange (Paar and Pelzl, 2010).	20
2.14	Tamanho das chaves recomendado pela ENISA na aplicação de vários algoritmos, * significa que o valor pode ser inferior dependendo da implementação (Weintraub and Blunden, Bill, 2013).	22
3.1	Abstração da proposta de solução.	25
3.2	Ciclo de vida de uma atualização FOTA.	26
3.3	Diagrama da arquitetura física da plataforma.	27
3.4	Arquitetura do dispositivo e respetivas ligações.	30
3.5	Diagrama de sequência referente ao caso de uso de atualização FOTA.	31
3.6	Diagrama de sequência relativo à execução do protocolo.	32
4.1	Diagrama de arquitetura e tecnologias usadas.	34
4.2	Estrutura do projeto em árvore.	35
4.3	Representação da estrutura da trama por níveis.	37
4.4	Diagrama de sequência do caso de uso de atualização FOTA após restrição de âmbito.	44
5.1	Página de conexões da interface <i>web</i>	46
5.2	Carregamento de um ficheiro exemplo via <i>drag-and-drop</i> através da interface <i>web</i>	47
5.3	Página de <i>log</i> da interface <i>web</i>	47

LISTA DE FIGURAS

5.4	Página de métricas da interface <i>web</i>	48
5.5	Registo das várias etapas da execução do protocolo FOTA.	50
5.6	Reenvio de um pacote de dados	51
5.7	Teste de validação da autenticação com IMEI desconhecido.	52
5.8	Validação de integridade através do CRC.	52
5.9	Adulteração de dados após aplicação de assinatura digital.	53
5.10	Assinatura de tramas com diferentes chaves RSA.	54
5.11	Utilização de recursos do servidor em vários casos de teste.	55
5.12	Duração da execução protocolo em diferentes casos de teste.	56
5.13	Visão geral da utilização de recursos nos cinco casos de teste.	57

Lista de Tabelas

3.1	Lista das principais características das alternativas analisadas	24
5.1	Resultados do teste de escala.	53
5.2	Consumo aproximado de recursos nos vários casos de teste.	54

LISTA DE TABELAS

Abreviaturas e Símbolos

AES	Advanced Encryption Standard
API	Application Programming Interface
AWS	Amazon Web Services
CAN	Controller Area Network
DL	Discrete Logarithm
ECC	Elliptic-curve cryptography
ECU	Electronic Control Unit
ENISA	European Union Agency for Network and Information Security
EUGDPR	EU General Data Protection Regulation
FOTA	Firmware Over The Air
GDPR	General Data Protection Regulation
GSM	Global System for Mobile communications
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IOT	Internet of Things
IT	Information Technology
NIST	National Institute of Standards and Technology (EUA)
OEM	Original Equipment Manufacturer
OTA	Over-the-air
OTAP	Over-the-air Provisioning
OTASP	Over-the-air Service Provisioning
PKI	Public Key Infrastructure
SOTA	Software Over The Air
TCP	Transmission Control Protocol
UI	User Interface
WSN	Wireless Sensor Network

Capítulo 1

Introdução

Num mundo cada vez mais conectado e em constante mutação tecnológica, emergem novos dispositivos, serviços e aplicações, os quais necessitam de ser atualizados ao longo do seu ciclo de vida. O processo de atualização é uma peça chave na evolução de um produto e hoje em dia é aplicado com uma frequência crescente, quer seja pela necessidade de introdução de novas funcionalidades, correção de vulnerabilidades ou para efeitos de manutenção.

1.1 Contexto

Este tema de dissertação insere-se na área de Sistemas Computacionais, com âmbito tecnológico ao nível de Sistemas Embebidos, abordando também conceitos de segurança e redes. A dissertação foi desenvolvida em conjunto com a empresa AddVolt.

Esta empresa desenvolve produtos para o sector automóvel, nomeadamente em veículos com carroçaria refrigerada. Os seus produtos permitem dotar um veículo de capacidades de geração e armazenamento de energia. Quer por via da instalação de um conjunto de painéis fotovoltaicos, de um conjunto de baterias ou de um sistema de travagem regenerativa. A energia acumulada é posteriormente usada para alimentar o sistema de refrigeração do veículo, traduzindo-se numa poupança de consumo de combustível significativa, resultando em benefícios ao nível económico e ambiental ([AddVolt, 2018](#)).

Existem duas abordagens no processo de atualização de um dispositivo, podendo este processo decorrer por via de intervenção física ou por via remota. Ambas as abordagens têm aplicação no mercado de acordo com as características do sistema em que são utilizadas. A complexidade do processo de atualização escala com o crescimento do número de dispositivos alvo de atualização, principalmente quando estes se encontram dispersos geograficamente e a sua localização varia em tempo real.

1.2 Motivação e Objetivos

Os produtos mencionados anteriormente possuem um dispositivo que interage com o sistema e envia informações para um servidor da forma periódica. Esses produtos são compostos por vários microcontroladores, os quais possuem *firmware* responsável pelo controlo do sistema e cumprimento das funções definidas pelos responsáveis da empresa. Ao longo do tempo vão sendo adicionadas funcionalidades e correções de falhas que resultam em novas versões de *firmware*, subsequentemente transferidas para os dispositivos.

Presentemente esta operação é realizada manualmente, o que resulta numa série de inconvenientes tanto para a empresa como para os seus clientes, uma vez que exige a deslocação de engenheiros da empresa ao local, a cada lançamento de uma nova versão de *firmware*.

Esta abordagem limita claramente a flexibilidade e escalabilidade do processo de atualização dos dispositivos, motivando a transição para um modelo de atualização remota recorrendo a programação *over-the-air* (OTA).

Em termos de objetivos, pretende-se analisar o estado atual de programação OTA tanto ao nível da indústria como a nível académico, a fim de desenhar e implementar um protótipo de uma plataforma capaz de gerir o lançamento de novas versões de *firmware* remotamente e com segurança. Permitindo ao utilizador carregar ficheiros de *firmware* bem como monitorizar o progresso do protocolo de transferência de dados.

Deve ter-se em conta que a adaptação da proposta de solução à infraestrutura da empresa existente constitui uma limitação, uma vez que implica seguir os seus padrões de desenvolvimento e o uso de tecnologias compatíveis, numa perspectiva de integração futura.

1.3 Metodologia

A metodologia posta em prática ao longo do período da dissertação, seguiu uma abordagem iterativa e incremental, com várias etapas de desenvolvimento: iniciando-se com um levantamento de requisitos da plataforma, seguida pelo estudo do estado da arte e da indústria, elaboração de uma proposta de solução, planeamento e desenho da arquitetura referente à proposta, implementação e testes.

A fase de implementação foi fragmentada em duas fases principais: desenvolvimento do protocolo de comunicação cliente-servidor e desenvolvimento da interface *web*. Cada uma das fases foi dividida em tarefas, normalmente correspondendo à implementação de uma funcionalidade específica, testada de forma individual e global a fim de manter a estabilidade da lógica implementada até ao momento. Nesta fase foi usado o sistema de controlo de versões *git*, associado a um repositório de código alojado na *cloud* na plataforma Bitbucket.

O período de testes à plataforma numa fase pós-implementação, foi fundamental para a recolha de métricas que permitiram avaliar o cumprimento dos requisitos definidos bem como retirar conclusões em relação à eficiência e robustez do protótipo implementado.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é analisado o estado corrente de programação OTA, mais concretamente na sua aplicação de atualização de *firmware* e *software* de dispositivos.

Pretende-se proporcionar uma visão geral sobre o que é programação OTA, quais os principais aspetos associados a atualizações OTA, terminologias associadas, bem como as principais aplicações na indústria e tendências na evolução desta tecnologia. Neste âmbito é ainda abordado o tema da criptografia e implicações de segurança no desenho de uma solução deste tipo.

2.1 Atualizações OTA

Antes de abordar a questão das atualizações OTA é necessário começar por contextualizar o que é programação OTA bem como as várias terminologias associadas a este conceito.

O termo OTA aplica-se a qualquer tipo de transmissão *wireless* de dados, por sua vez programação OTA refere-se a sinais usados para programar, ativar, configurar e atualizar dispositivos de *hardware* (Techterms, 2016). Esta técnica surgiu no decorrer do desenvolvimento de métodos que permitiam a distribuição remota de dados tais como ficheiros de configuração, chaves de encriptação, *software* etc. de um ponto centralizado para um conjunto de clientes. A adoção deste mecanismo de atualizações remotas acabou por tornar-se um padrão nas empresas pois representava uma maior flexibilidade no processo global de atualização dos seus dispositivos.

Existem outros conceitos relacionados tais como *over-the-air Provisioning* (OTAP) que consiste numa técnica de gestão de pontos de acesso (AP) de forma a conseguir fornecer dados de configuração remotamente (Cisco, 2011). Outro termo mais usado no âmbito da indústria dos dispositivos móveis é *over-the-air Service Provisioning* (OTASP), uma técnica que permite ativar serviços no dispositivo móvel sem necessidade de acesso físico ao mesmo (Randall A. Snyder, 2001).

A segurança é um ponto importante a ter em consideração no desenvolvimento de protocolos de atualização OTA. Os protocolos devem seguir padrões que garantam características robustas ao

nível da integridade, confidencialidade e autenticidade nos dados transmitidos. (Eystein Stenberg, 2017). Estes aspetos são abordados na secção 2.2 relativa ao tema da Criptografia. Para além destes conceitos existem ainda outras questões a ter em conta na aplicação de programação OTA, tais como:

- **Overhead no processamento** - os recursos computacionais dos dispositivos alvo são por norma reduzidos, nesse sentido a *performance* torna-se um ponto crítico no processo de atualização OTA, o qual não deverá interferir com o funcionamento normal do dispositivo.
- **Consumo de largura de banda** - o protocolo utilizado deve ser leve e considerar a infraestrutura de rede a que o dispositivo se encontra ligado bem como o consumo de tráfego usado no seu funcionamento normal.

A combinação de todos estes elementos mencionados faz escalar a complexidade no planeamento e desenvolvimento de uma plataforma ligada a programação OTA, motivando a resistência inicial da indústria na adoção deste paradigma.

2.1.1 Atualizações OTA em dispositivos móveis

Hoje em dia os dispositivos móveis contêm quantidades enormes de software, maioritariamente *firmware*, que permitem dar suporte a funcionalidades cada vez mais avançadas (Sharon Peleg, 2007).

Em 2006, a Open Mobile Alliance (OMA), principal organização no que diz respeito ao desenvolvimento de *standards* para indústria dos telemóveis (Ltd., 2017), lançou o Firmware Update Management Object (FUMO), uma especificação orientada à atualização de *firmware over-the-air* em dispositivos móveis.

Na figura 2.1 pode observar-se um diagrama de uma representação alto nível do processo base de *download* de uma atualização, ativada por um sistema externo que comunica com um *Device Management (DM) server* o qual vai iniciar o protocolo definido na especificação FUMO.

Em termos de implementação na indústria, a abordagem apresentada na figura anterior traz vantagens em termos de modularização do sistema, tendo um servidor dedicado à execução do protocolo de atualização, o qual recebe comandos de um módulo externo de uma camada superior que vai desencadear a distribuição de atualizações de *firmware* nos dispositivos.

Esta especificação surge como resposta à tendência do mercado, deixando de exigir que os consumidores ou fabricantes iniciem atualizações através de contacto direto, marcando assim um ponto de viragem no que diz respeito ao modelo de atualização dos telemóveis (Alliance, 2012), resultando como prática padrão o suporte a FOTA nos dispositivos móveis por parte dos fabricantes.

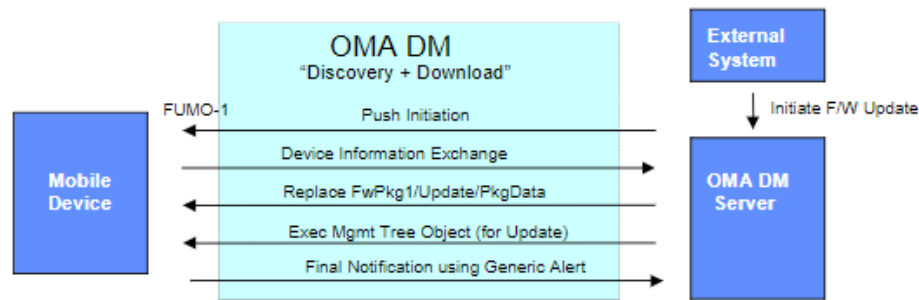


Figura 2.1: Diagrama de sequência base do *download* de um pacote de atualização (Open Mobile Alliance Ltd., 2007),

2.1.2 Atualizações OTA em WSNs

Em 2020 estima-se que mais de 50 mil milhões de dispositivos estejam ligados à Internet (Evans, 2011). Parte desta revolução deve-se sem dúvida ao *boom* do conceito de *Internet of Things* (IoT). O número de dispositivos conectados à Internet cresceu de forma abrupta, desencadeando o aparecimento de novos modelos de negócio nesta área, como resultado do forte investimento neste sector. Por sua vez o surgimento destes novos modelos de negócio atrai mais investimento em termos de investigação e desenvolvimento de novas plataformas e serviços (libellium, 2012).

Neste âmbito emerge o conceito de *Wireless Sensor Network* (WSN), que é definido como uma rede de sensores (nós) conectados sem-fios, auto-configurados e sem necessidade de uma infraestrutura, normalmente utilizados para monitorização de condições físicas ou ambientais (Matin, 2006). Estes dispositivos comunicam entre si com intuito de propagarem a informação recolhida até um determinado *gateway* que irá encaminhar os dados para a plataforma de gestão da rede. É esperado que estas redes sejam implantadas a longo prazo o que torna inevitável a necessidade de realizar atualizações remotas de *software* ou *firmware*, uma vez que normalmente os nós são inacessíveis ou são demasiado numerosos para serem acedidos fisicamente.

Surge também neste contexto a necessidade de recorrer a atualizações OTA tirando partido das propriedades de disseminação de informação entre os nós da rede. Existem várias propostas de protocolos nesta área, o SenSeOP (Aschenbruck et al., 2012) é um desses exemplos de protocolos de aplicação de programação OTA em WSNs com algum grau de heterogeneidade ao nível do *hardware* dos sensores, permitindo assim selecionar grupos de nós alvo de atualizações OTA via *multicast*. Este protocolo tem em atenção a questão da segurança aplicando métodos que visam evitar ataques de reprogramação de nós, ataques *replay* e DoS. Para tal é usada criptografia assimétrica, guardando apenas a chave pública em cada nó, incluindo também assinaturas digitais baseadas em ECC e contadores de versões, a fim de manter a integridade e autenticidade dos dados.

Tem havido um crescimento no número de empresas que dispõem de soluções ligadas ao fabrico de componentes e dispositivos destinados a IoT. Um desses casos é a Libelium, empresa que se insere na categoria de fabricante de *hardware*, ligada a projetos de impacto ambiental e cooperação no desenvolvimento de *smart cities*. Desenvolveu uma plataforma que possui uma implementação de programação OTA sem-fios, compatível com os módulos 3G, 4G, GPRS e Wi-Fi (Libelium, 2017). A figura 2.2 ilustra a plataforma referida, que disponibiliza um servidor FTP o qual é questionado pelos dispositivos (sensores) sobre a existência de novas versões de *firmware*, via *polling*, que por sua vez realiza a transferência após validar os parâmetros recebidos.

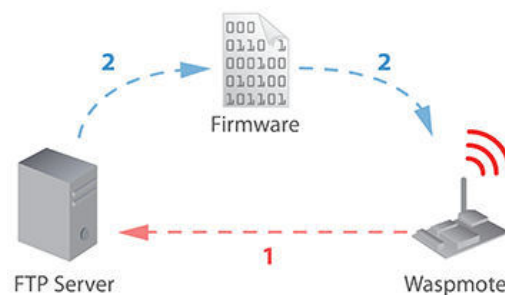


Figura 2.2: Representação alto nível do processo de atualização OTA de um sensor (Libelium, 2017).

A interação com o protocolo é realizada através de uma interface que permite carregar ficheiros binários, listar os nós activos, obter informações do sistema, entre outras funcionalidades. Os principais benefícios advindos da utilização desta plataforma são:

- Actualização ou alteração de *firmware* sem necessidade de acesso físico.
- Actualização do *firmware* consultando um servidor FTP, o que ajuda a fazer uma boa manutenção da bateria.
- Transferência do novo *firmware* em poucos minutos.

2.1.3 Actualizações OTA na indústria automóvel

Nas últimas décadas os componentes mecânicos dos automóveis vêm sendo substituídos por componentes eletrónicos e a tendência é crescente, onde se estima que em 2030 cerca de 50% do custo total de um carro seja derivado dos seus componentes eletrónicos (Statista, 2017), tal como pode ser observado na figura 2.3.

O surgimento do Engine Control Unit (ECU), componente eletrónico responsável pelo controlo do funcionamento do motor, veio revolucionar a indústria automóvel e abrir as portas a uma demanda crescente por componentes eletrónicos na indústria automóvel. Os ECUs modernos fazem uso de um microprocessador, formado por um componente de *hardware* e *firmware* associado, capaz de analisar e processar *inputs* em tempo real. Estima-se que hoje em dia os veículos

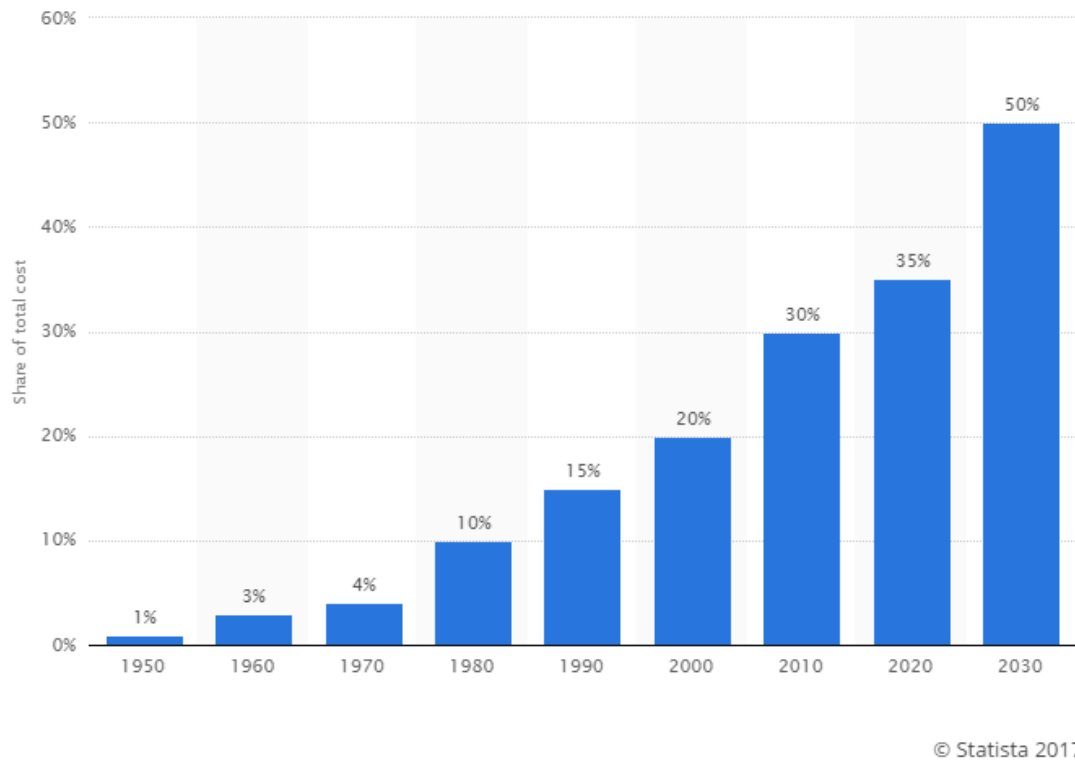


Figura 2.3: Diagrama do custo compartilhado do *Engine Control Unit* (ECU) em relação ao custo total do automóvel (Statista, 2017).

produzidos estejam equipados com uma gama de 50 a 70 ECUs embebidos (Daimi et al., 2016), executando diferente *firmware*, responsável por diversas funcionalidades do veículo, para além do controlo do funcionamento do motor. Os vários ECUs necessitam de ser atualizados ao longo do tempo, para remoção de *bugs*, melhorar a segurança e introdução de novas funcionalidades. A atualização tradicional dos ECUs é feita por via manual, no entanto este paradigma está a mudar para uma abordagem OTA. O sucesso da Tesla no uso de atualizações OTA ligado ao campo dos automóveis autónomos obrigou as OEMs concorrentes a reconhecer que este é o caminho a seguir no futuro do sector automóvel, prevendo-se que em 2022 exista um volume de 203 milhões de veículos a saírem de fábrica com compatibilidade com acesso OTA (Research, 2016).

No mercado automóvel existem cada vez mais empresas ligadas ao desenvolvimento de componentes para ECUs e de soluções com aplicação de programação OTA. Um destes exemplos é a Infineon, uma empresa dedicada ao fabrico de semicondutores, com várias aplicações na indústria, nomeadamente na automóvel (Infineon, 2017), que disponibiliza *hardware* dedicado a *software over-the-air* (SOTA) podendo ser integrado num sistema automóvel, como ilustrado na figura 2.4. Os três componentes representados combinam funcionalidades de acesso à rede móvel, autenticação e encriptação, transferência, armazenamento e instalação de *software*. Este é um exemplo das abordagens de programação OTA existentes na indústria automóvel, fornecendo um

panorama geral no que diz respeito ao avanço do paradigma OTA na indústria automóvel.

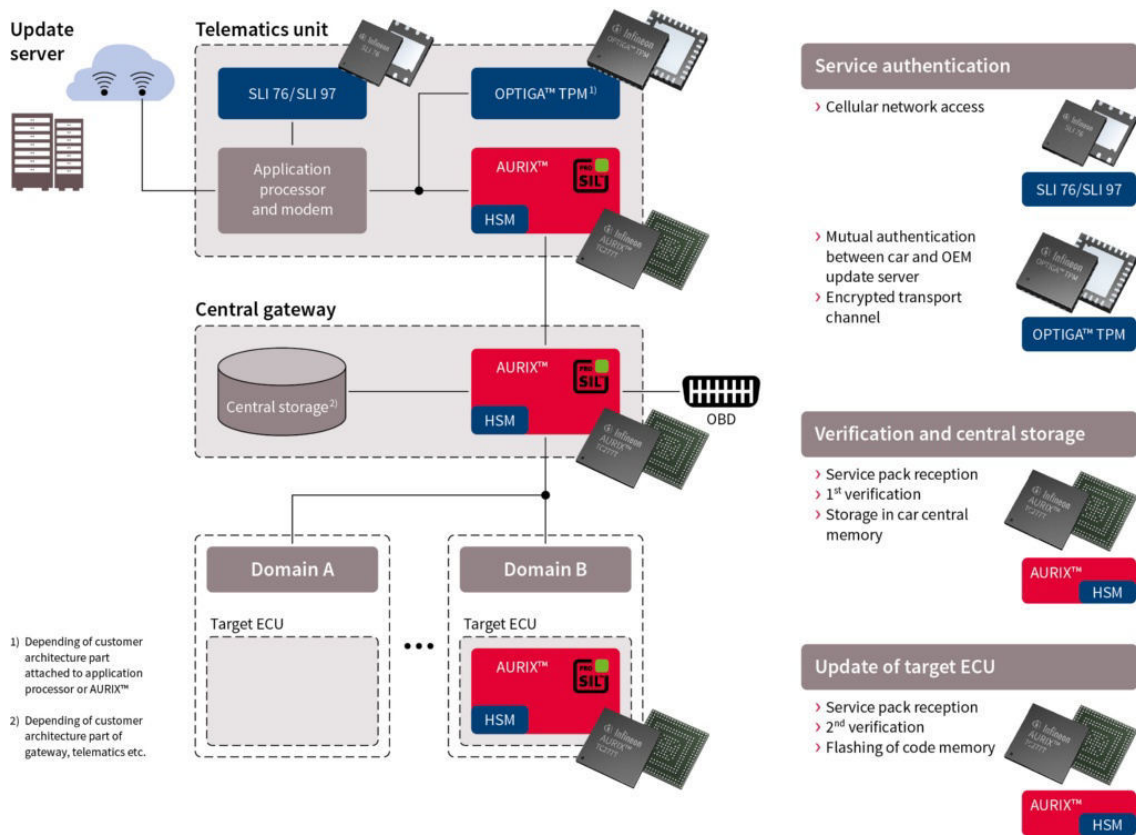


Figura 2.4: Esquema de integração de produtos da Infineon num sistema automóvel dedicados a operações de SOTA (Infineon, 2017).

2.2 Criptografia

A introdução de computadores como equipamentos de processamento de dados contribuiu em massa para o eficiente armazenamento e processamento dos dados. Esta automatização levou à necessidade da criação de ferramentas que protegessem as informações armazenadas. O aparecimento de redes de comunicação e sistemas distribuídos introduziu uma maior necessidade de proteção de informação, nomeadamente em termos de segurança na transferência dos dados (Francia et al., 2017).

As pequenas e médias empresas (PMEs) estão atualmente a dominar o panorama europeu, constituindo o *backbone* da economia europeia. A fim de atingirem os seus objetivos, as PME's vêm demonstrando uma dependência crescente de utilização de sistemas de IT, incluindo-os nas suas infraestruturas ou recorrendo a serviços de terceiros. De acordo com o Regulamento Geral sobre a Proteção de Dados (RGPD) o termo segurança abrange igualmente a confidencialidade, integridade e disponibilidade e deve ser considerada segundo um modelo baseado no risco, ou seja,

quanto maior o risco, mais rigorosas devem ser as medidas aplicadas a fim de gerir o risco (ENISA, 2016).

Focando em alguns dos pontos expostos na figura 2.5, mencionados anteriormente na introdução às características fundamentais na abordagem a programação OTA, pode definir-se:

- **Confidencialidade** - a informação não deve ser perceptível em caso de interceção, conseguido através do uso de algum mecanismo de encriptação, onde os intervenientes apenas podem decifrar os dados se tiverem na sua posse a chave de encriptação (Francia et al., 2017).
- **Integridade** - deve assegurar-se a precisão e a completude dos dados ao longo de todo o seu ciclo de vida, garantindo que os dados não são modificados de forma não autorizada ou não detetada. Esta característica é normalmente alcançada pelo intermédio da aplicação de funções de *hashing* (Francia et al., 2017).
- **Autenticidade** - os dados recebidos devem ser provenientes de uma fonte segura e que verifique os privilégios necessários para transmitir a informação.

Para cumprir com estas propriedades de segurança é necessário a aplicação de mecanismos criptográficos, tais como *hashing* e encriptação. A combinação destes mecanismos permite gerir informação sensível garantindo as propriedades de segurança necessárias. Os algoritmos criptográficos são normalmente classificados de acordo com a natureza de chaves utilizadas para encriptação e respetiva desencriptação, resultando em três classes de algoritmos, funções de *hashing*, algoritmos de chave secreta (i.e., encriptação simétrica) e algoritmos de chave pública (i.e., encriptação assimétrica) (Kessler, 2017).

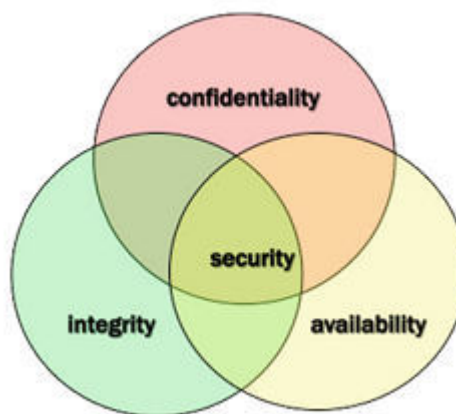


Figura 2.5: Segurança na intersecção da confidencialidade, integridade e disponibilidade.

2.2.1 Hashing versus encriptação

É importante definir estes dois termos como sendo dois processos distintos. Ao encriptar um bloco de dados, estão a realizar-se operações que transformam os dados alterando o seu valor

original, no entanto o bloco gerado (encriptado) pode posteriormente ser desencriptado, voltando assim ao seu estado original. O tipo de encriptação pode ser simétrica ou assimétrica, sendo estes temas abordados com mais detalhe nas próximas secções.

Por outro lado, *hashing* corresponde à aplicação de um algoritmo matemático que mapeia dados de tamanho arbitrário para uma cadeia de valor fixo. Ao aplicar *hashing* a um bloco de dados, não existe modificação do mesmo, apenas é gerado um valor de *hash* com base na informação do bloco. Não existe correspondência entre os dados originais e o valor de *hash*, sendo inviável reverter o processo de *hashing*, isto é, não conseguimos obter a informação original a partir do valor de *hash* resultante, tratando-se de um método denominado *one-way* (Jr. and Antonakos, 2009). É com base no facto de uma função de *hashing* ser *one-way* (i.e., não reversível) que se estabelece a maior distinção em relação à encriptação.

Podem definir-se dois tipos de funções de *hashing*, as dedicadas que correspondem a algoritmos desenhados especificamente para servir como funções de *hashing* e as baseadas em *block cipher* (ex. SHA-2). As funções de *hashing* conseguem processar dados com comprimento arbitrário resultando no final num *hash* de tamanho fixo. Isto é conseguido através da segmentação dos dados de entrada em blocos de tamanho igual, que por sua vez são processados sequencialmente pela função de *hashing*, a qual possui uma função de compressão no *core* da sua lógica. Esta abordagem de processamento iterativo é conhecida como *Merkle–Damgård construction* (Paar and Pelzl, 2010) e está presente em diversos algoritmos como MD5, a família de algoritmos SHA (exceto SHA-3), entre outros menos utilizados atualmente.

Exemplo de aplicação de *hashing* a uma palavra recorrendo ao algoritmo MD5:

$$\text{MD5("Alice")} = 64489c85dc2fe0787b85cd87214b3810$$

O comprimento do valor de *hash* gerado varia de acordo com a função usada, podendo ser diferente do comprimento dos dados de entrada. Neste caso, aplicou-se o algoritmo MD5 que usa uma função que gera *hashes* de 128 bits (16 bytes), cujo valor é representado por 32 caracteres hexadecimais. Mesmo com uma pequena alteração, por exemplo se for adicionado um carácter à palavra, verifica-se que o valor de *hash* gerado é completamente diferente.

$$\text{MD5("Alice1")} = b27fad39cc2bfa5c2fb3aefbc1ee1fa1$$

Uma função de *hashing* criptográfica $H(x)$, deve ter em conta os requisitos apresentados na figura 2.6. Se uma função de *hashing* cumpre com as primeiras cinco propriedades observadas na tabela da imagem, considera-se como sendo uma função de *hashing* fraca, se verificar também a sexta propriedade (*collision resistant*) passa a ser considerada como uma função de *hashing* forte.

Pode usar-se *hashing* para criação de *hash chains*, que consiste na aplicação sucessiva de *hashing*, inicialmente a uma dada *string* cuja *hash* resultante serve como entrada do *hashing* seguinte e assim sucessivamente, este conceito pode ser usado na proteção de *passwords* e também para garantir o não-repúdio, isto é o autor de uma mensagem não consegue negar a autoria da

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x , making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h , it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x , it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Figura 2.6: Requisitos para uma função de *hashing* criptográfica (Stallings, 2013).

mesma (e.g., no caso de querer manter-se um registo cronológico de determinados blocos de informação recebidos).

Outro exemplo da aplicação de *hashing* é em *Hash-based Message Authentication Code* (HMAC), um tipo específico de MAC (método que dada uma mensagem e uma chave simétrica retorna uma *tag* que pode ser adicionada a uma mensagem com o objectivo de garantir integridade e autenticidade da mesma). O uso de HMAC confere maiores propriedades segurança e maior eficiência computacional, no entanto não garante o não-repúdio (Paar and Pelzl, 2010).

	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + digital signature	yes	yes	yes*
Intrusion detection and virus detection		yes	
Hash + symmetric encryption			
One-way password file	yes		
MAC	yes	yes	yes*

*Resistance required if attacker is able to mount a chosen message attack

Figura 2.7: Propriedades de resistência de uma função de *hashing*, necessárias em mecanismos de manutenção de integridade de dados (Stallings, 2013).

Para além da verificação de integridade de ficheiros ou mensagens, existem outras aplicações

de *hashing* como: armazenamento/verificação de *passwords*, identificação de ficheiros ou dados (ex. assinaturas digitais), *proof-of-work* (ex. Bitcoin). Na figura 2.7 são apresentadas propriedades de resistência presentes em algumas abordagens à manutenção de integridade de dados.

Tal como no caso dos algoritmos de encriptação, existem duas categorias de ataques a funções de *hashing*: ataques de força-bruta, geralmente designados de ataques *brute-force*, e ataques de *cryptanalysis*. Em relação aos ataques de força-bruta estes estão divididos em ataques *Preimage*, *Second Preimage* e *collision resistant*. Ataques criptoanalíticos em funções *hashing* procuram expor uma determinada propriedade ou estado interno do algoritmo a fim de ser usada para realizar um ataque não exaustivo. Um algoritmo de *hashing* ideal deve exigir um esforço criptoanalítico maior ou igual ao de um ataque comum de força-bruta (Stallings, 2013).

O poder computacional segue uma tendência crescente com o surgimento de supercomputadores cada vez mais poderosos, que combinado com a atual evolução da computação quântica, que existe hoje em dia como serviço disponibilizado ao público por parte da indústria, e.g., Microsoft (Microsoft, 2017), fez soar os alarmes dos criptógrafos em termos da fragilidade dos algoritmos criptográficos existentes face ao crescendo computacional. Ao longo das últimas décadas vêm sendo expostas vulnerabilidades em vários algoritmos de *hashing* criptográfico, nomeadamente em algoritmos da classe MD e SHA. Na figura 2.8 pode observar-se uma lista de ataques registados e apontados numa publicação feita pela Microsoft em 2005.

hash	attack			
	author	type	complexity	year
MD4	Dobbertin	collision	2^{22}	1996
	Wang et. al	collision	2^8	2005
MD5	dan Boer & Bosselaers	pseudo-collision	2^{16}	1993
	Dobbertin	free-start	2^{34}	1996
	Wang et. al	collision	2^{39}	2005
SHA-0	Chabaud & Joux	collision	2^{61} (theory)	1998
	Biham & Chen	near-collision	2^{40}	2004
	Biham et. al	collision	2^{51}	2005
	Wang et. al	collision	2^{39}	2005
SHA-1	Biham et. al	collision (40 rounds)	very low	2005
	Biham et. al	collision (58 rounds)	2^{75} (theory)	2005
	Wang et. al	collision (58 rounds)	2^{33}	2005
	Wang et. al	collision	2^{63} (theory)	2005

Figura 2.8: Ataques em funções *hashing* padrão (Mironov, 2005).

Recentemente, em 2017, foi provado em sentido prático que o mundialmente usado algoritmo SHA-1 é vulnerável a ataques de colisão (*collision attacks*) (Stevens et al., 2017) confirmando os bastantes indícios já publicados em diversos livros e *papers* desde o início do milénio. Apesar de ainda não existirem ataques publicados relativamente ao algoritmo SHA-2, o atual padrão (SHS)

adotado pelo National Institute of Standards and Technology (NIST) é o SHA-3, uma vez que o SHA-2 partilha a mesma estrutura e operações matemáticas do seu predecessor (SHA-1) e portanto se o seu predecessor é vulnerável a ataques, teoricamente SHA-2 também será (Stallings, 2013).

Concluindo, há que ter em conta as lições aprendidas até ao momento das quais resultam alguns aspetos que devem ser considerados ao utilizar funções de *hashing*. Devem ser usadas funções que retornem ao final um *hash* de pelo menos 160 bits, de modo a suportar ataques do tipo colisão. Se considerarmos uma perspetiva de longo termo a melhor opção é optar por uma função que retorne um *hash* de pelo menos 256 bits. Os algoritmos MD5 e SHA-1, usados extensivamente em todo o globo, são inseguros e o seu uso deve ser evitado.

2.2.2 Encriptação simétrica

A encriptação simétrica, igualmente designada de chave-secreta ou chave-simétrica, é um mecanismo bastante direto que se encontra na base da criptografia. A sua introdução foi motivada pela necessidade de garantir autenticidade e confidencialidade numa comunicação entre várias partes através de um canal inseguro. Este método foi desenhado com intuito de tornar a informação contida na mensagem transmitida, entre as várias partes, impercetível em caso de interceção de terceiros.

A encriptação simétrica consiste na cifragem de uma determinada mensagem, utilizando para tal uma chave que é partilhada pelas partes confiadas, a qual é também usada para descriptar a mensagem (Stevens et al., 2012). Uma vez encriptada, está garantida a sua confidencialidade.

A encriptação simétrica também proporciona um certo grau de autenticidade se se assumir que apenas as partes intervenientes, emissor e recetor, possuem a chave secreta, o que de qualquer forma não impede que terceiros possam alterar ou repetir a mensagem encriptada. O diagrama mostrado na figura 2.9 apresenta, de uma forma geral, o funcionamento da encriptação simétrica.

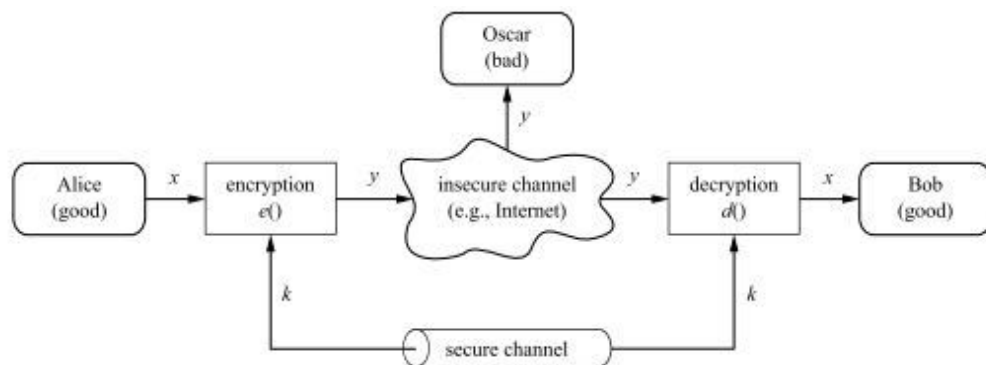


Figura 2.9: Encriptação simétrica (Paar and Pelzl, 2010).

A transmissão da chave entre as várias partes deve ser realizada através de um canal seguro e o algoritmo usado deve ser conhecido publicamente, visto que a única forma de garantir que um

algoritmo de encriptação é forte, ou seja é difícil de quebrar, passa por tê-lo disponível para que outros criptógrafos o analisem. A encriptação simétrica pode ser vulnerável a ataques de força-bruta (*Exhaustive Key Search*) e a ataques de *cryptanalysis* que usam uma tabela estatística com as frequências das letras do alfabeto de acordo com a língua alvo (Paar and Pelzl, 2010).

O uso da especificação AES nas suas variantes 128, 192 e 256 bits, tornou-se padrão a nível global no que diz respeito a implementações de sistemas de encriptação simétrica. Existem até instruções AES ao nível do *hardware* presentes em processadores Intel fabricados a partir de 2010 (Gueron, 2010). Apesar das chaves mais longas (192 e 256 bits) garantirem um maior nível de segurança, apresentam também reduções de desempenho relativamente à variante de 128 bits, na ordem dos 20, 40% respetivamente. Em termos de vulnerabilidades, foi demonstrado que o AES pode ser vulnerável a ataques algébricos, no entanto estes ainda não são computacionalmente viáveis num futuro próximo (Weintraub and Blunden, Bill, 2013). De acordo como o NIST, no caso da encriptação simétrica, o uso de uma chave com um tamanho, e.g., de 128 bits é considerado como sendo seguro até 2040 (Giry, 2017).

2.2.3 Encriptação assimétrica e assinaturas digitais

A encriptação assimétrica, também designada de encriptação de chave pública é um desenvolvimento recente em relação à criptografia simétrica. O seu aparecimento foi motivado pela existência de algumas limitações numa abordagem de chave-simétrica, tais como o problema da distribuição de chaves, onde a ligação entre o emissor e recetor é insegura, logo a chave não podendo ser enviada através desse meio. Outro problema reside no volume de chaves em circulação, no caso de uma rede de 2000 pessoas, por exemplo, teriam de ser gerados 2000 x 2000 pares de chaves, resultando em quatro milhões de pares de chaves que têm de ser gerados e transmitidos por via de canais seguros (Paar and Pelzl, 2010). No entanto é necessário ressaltar que a criptografia assimétrica não vem substituir a criptografia simétrica, o *overhead* computacional das implementações correntes implica que a encriptação simétrica continue a ser utilizada.

Um algoritmo de chave pública consiste num algoritmo criptográfico que usa duas chaves relacionadas mas diferentes, designadas de chave pública e chave privada. Baseia-se na característica de que é computacionalmente inviável determinar a chave de descriptação conhecendo apenas o algoritmo usado e a chave de encriptação. Estas chaves assimétricas são usadas em operações complementares como encriptação e descriptação ou na geração e verificação de assinaturas.

O funcionamento destes algoritmos depende de um certificado de chave pública, que corresponde a um documento digital assinado com a chave privada de uma Autoridade Certificadora (CA), que liga um dado subscritor a uma chave pública, correspondendo à respetiva chave privada. Se a chave pública for usada para encriptar a mensagem então para descriptar terá de ser usada a chave privada correspondente e vice-versa. Também é fundamental definir a terminologia de Infraestrutura de Chave Pública (PKI) que fundamentalmente reflete um conjunto de políticas,

plataformas de servidor, *software* e estações de trabalho, usados com o propósito de gerir e administrar certificados e pares de chaves (pública e privada) (Stallings, 2013).

Em linhas gerais o funcionamento de um algoritmo de encriptação assimétrica passa pelas seguintes etapas:

1. Cada utilizador gera um par de chaves que vão ser usadas na encriptação e desencriptação das mensagens.
2. Cada utilizador expõe uma das chaves num registo acessível por ambas as partes, quanto mais público melhor uma vez que torna o processo mais confiável, estas chaves passam a ser designadas como chaves públicas. O par de cada chave pública, é mantido em privado, passando a ser designado como chave privada.
3. Se um utilizador pretender enviar uma mensagem, acede ao registo, obtém a chave pública do destinatário e usa-a para encriptar a mensagem.
4. O destinatário ao receber a mensagem, desencripta-a usando a sua chave privada. Desta forma apenas o destinatário da mensagem pode decifrar o conteúdo da mensagem uma vez que é o único que possui a chave privada.

Na figura 2.10 apresenta-se um diagrama que ilustra o funcionamento geral da criptografia de chave pública. O primeiro processo (a) corresponde aos passos descritos anteriormente, o segundo processo (b) corresponde ao funcionamento de uma assinatura digital (garante não-repúdio). Neste caso o emissor encripta a mensagem com a sua chave privada e envia-a ao destinatário, este ao recebê-la desencripta-a usando a chave pública do emissor. O facto do destinatário conseguir desencriptar a mensagem usando a chave pública do emissor verifica que apenas o emissor poderia ter enviado a mensagem uma vez que só ele possui a chave privada. Desta forma apenas é impossível modificar a mensagem sem ter acesso à chave privada do emissor, portanto a mensagem tem garantia de autenticação da fonte e de integridade dos dados (Stallings, 2013).

Existem fundamentalmente três "famílias" de algoritmos de chave pública que seguem os seguintes esquemas:

- **Fatorização de inteiros** - esquema baseado no facto de ser difícil fatorizar inteiros grandes. Exemplo: RSA (Rivest–Shamir–Adleman).
- **Logaritmo discreto** - assentam no problema do logaritmo discreto em espaços finitos, isto é, baseiam a sua segurança no pressuposto de que não existe uma solução eficiente para o problema do logaritmo discreto quando aplicado a determinados grupos. Exemplo: Digital Signature Algorithm (DSA), Diffie-Hellman Key Exchange (DHKE).
- **Curva elíptica** - um esquema baseado na generalização do problema do logaritmo discreto em curvas elípticas. Exemplo: Elliptic Curve Diffie–Hellman key exchange (ECDH).

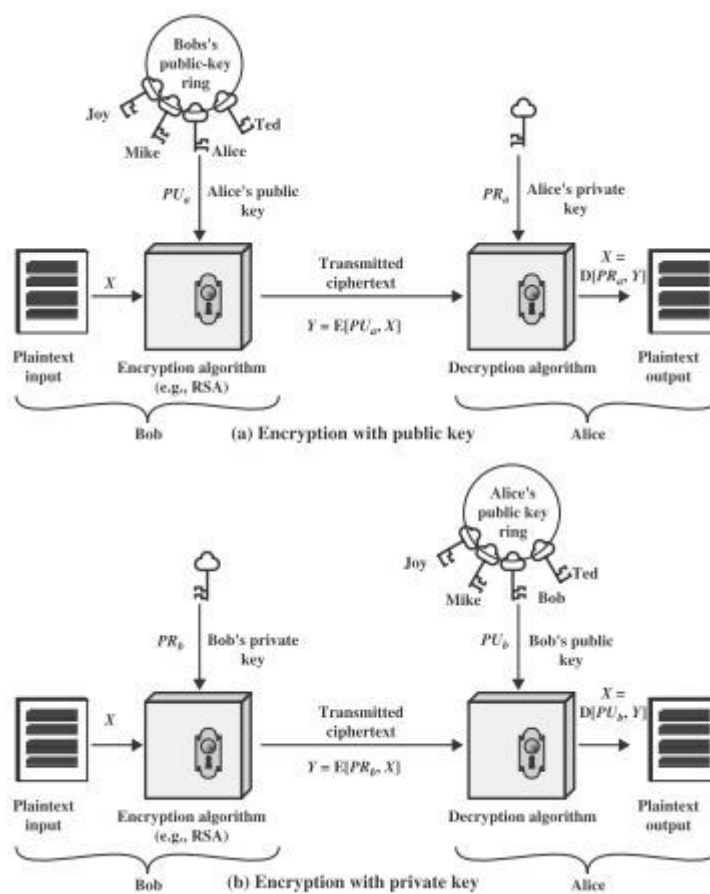


Figura 2.10: Encriptação assimétrica. Processo (a) encriptação com chave pública, (b) encriptação com chave privada (Stallings, 2013).

Na figura 2.11 ilustra-se uma tabela com as aplicações dos algoritmos referidos na lista anterior. Os algoritmos presentes na tabela anterior são considerados como sendo fortes, residindo a sua principal ameaça no poder computacional quântico. A computação quântica levanta implicações em termos de segurança resultando numa preocupação a nível de paradigma. Preocupação esta que, num futuro próximo, obrigará a que ocorra uma revisão da arquitetura e funcionamento destes algoritmos e da criptografia em geral, focando em temas como a Criptografia pós-quântica (Augot et al., 2015).

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Figura 2.11: Aplicações de encriptação assimétrica (Stallings, 2013).

2.2.4 Métodos de transferência de chaves

Nas próximas sub-seções descrevem-se dois métodos de acordo e transferência de chaves, o Diffie-Hellman Key Exchange (DHKE), método convencional, e o Elliptic Curve Diffie-Hellman Key Exchange (ECDHKE), que é mais recente. Ambos os métodos têm como objetivo estabelecer uma chave secreta partilhada (simétrica) através de um canal inseguro e podem ser subdivididos em dois protocolos, o protocolo de configuração, também designado de Diffie-Hellman *Set-up* e o protocolo principal. A principal diferença de funcionamento dos métodos encontra-se na fase inicial correspondente ao protocolo de configuração, que será detalhado nas próximas secções.

2.2.4.1 Diffie-Hellman Key Exchange (DHKE)

Este protocolo permite que dois intervenientes, comunicando numa canal inseguro, consigam derivar uma chave secreta comum. Como referido no final da secção anterior, este protocolo é um dos exemplos da aplicação do problema do Logaritmo Discreto (DL). Protocolos como Secure Shell (SSH), Transport Layer Security (TLS) e Internet Protocol Security (IPSec) fazem uso desta técnica de troca de chaves (Paar and Pelzl, 2010).

Esta técnica é sustentada na ideia de que a exponenciação em $\mathbb{Z}_p^*$, com p primo é uma função de sentido único, ou seja é uma função que é computacionalmente fácil de calcular para qualquer entrada (qualquer valor do seu domínio), mas computacionalmente difícil de inverter dada a imagem de uma entrada aleatória. É também sustentada pela propriedade comutativa da exponenciação (Paar and Pelzl, 2010), de acordo com a expressão seguinte:

$$k = (\alpha^x)^y \equiv (\alpha^y)^x \pmod{q}$$

Onde o segredo comum é representado pelo resultado da operação anterior, podendo ser usado como a chave de sessão entre as duas entidades. O funcionamento do protocolo é demonstrado na figura 2.12.

O protocolo de configuração consiste em três passos:

- Escolha de um primo q , normalmente grande.
- Escolha de um inteiro (gerador) $\alpha \in \{2, 3, \dots, q-2\}$.
- Comunicação dos parâmetros q e α (também conhecidos como parâmetros de domínio).

Operações no protocolo principal:

- As duas partes, representadas pela Alice (A) e pelo Bob (B), geram inicialmente uma chave privada, $X_A < q$ e $X_B < q$ respetivamente.
- De seguida ambos geram uma chave pública:

$$(a) Y_A = \alpha^{X_A} \pmod{q}$$

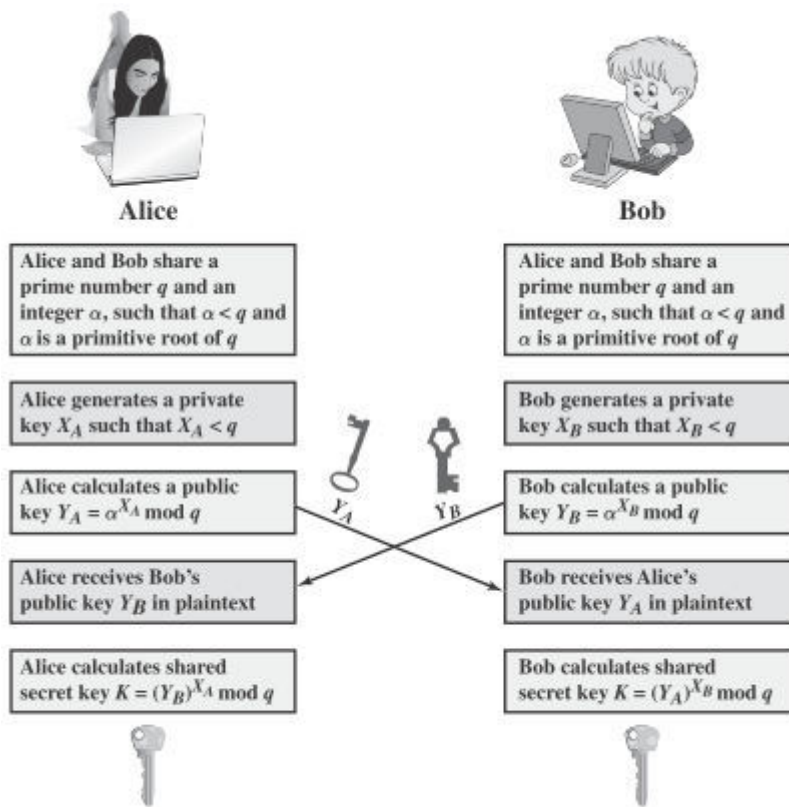


Figura 2.12: Diffie-Hellman Key Exchange (Stallings, 2013).

(b) $Y_B = \alpha^{X_B} \bmod q$

3. Partilham entre eles a sua chave pública.

4. Calculam a chave secreta comum (segredo) como resultados das operações:

(a) $K_A = Y_B^{X_A} \bmod q$

(b) $K_B = Y_A^{X_B} \bmod q$

Desta operação resulta uma chave secreta partilhada pela Alice e pelo Bob, de acordo com as propriedades seguintes:

- A Alice calcula $Y_B \equiv (\alpha^{X_B})^{X_A} \equiv \alpha^{X_A X_B} \bmod q$
- O Bob calcula $Y_A \equiv (\alpha^{X_A})^{X_B} \equiv \alpha^{X_A X_B} \bmod q$
- Logo, $K \equiv \alpha^{X_A X_B} \bmod q$

No decorrer do protocolo apenas são expostos os parâmetros primo (q), inteiro (α) e chaves públicas (Y_A e Y_B), mantendo-se sempre reservado o valor das chaves privadas (X_A e X_B) e da chave secreta (K). À chave obtida pode ser aplicada uma função de *hashing* com o intuito de remover bits "fracos" ou para a obtenção de uma chave com um tamanho definido. A chave secreta resultante

pode ser usada para estabelecer uma comunicação segura entre as duas entidades, podendo por exemplo ser aplicada na encriptação simétrica de informação com o uso de AES.

A segurança desta técnica de troca de chaves prende-se com a facilidade do cálculo de operações do tipo exponencial módulo de um primo e com a dificuldade de calcular logaritmos discretos, isto é, um atacante que pretenda determinar a chave do Bob teria de calcular:

$$X_B = d \log_{\alpha, q}(Y_B), \text{ em que } d \log \text{ representa logaritmo discreto}$$

Para números primos grandes torna-se inviável efetuar este tipo de cálculo, inviabilizando a abordagem "força bruta" na tentativa de reverter o processo. Para garantir um nível de segurança a médio prazo, é recomendada a escolha de um primo com pelo menos 2048 bits (Giry, 2017) conferindo uma complexidade de ataque de 2^{112} . Algumas considerações de segurança a reter são de que deverá ser usado um canal de comunicação seguro, caso contrário esta técnica será suscetível a ataques do tipo *man-in-the-middle* (MITM) (Stallings, 2013). Um canal de comunicação seguro consiste num método de transferir dados sem o risco de interceção e adulteração de terceiros. É normalmente conseguido aplicando protocolos criptográficos, e.g. SSL, TLS, que encriptam os dados transmitidos a fim de garantir a sua confidencialidade e integridade.

Existem variantes desta técnica, nomeadamente a Ephemeral Diffie-Hellman Key Exchange (EDHKE). Esta variante é interessante uma vez que gera chaves únicas por sessão que são descartadas no final da sessão, dotando a técnica de uma propriedade denominada de *perfect forward secrecy* (PFS), isto é, o facto de uma sessão ser comprometida não implica que sessões anteriores ou futuras sejam comprometidas. É por este facto considerada como técnica recomendada no melhoramento de serviços que façam uso de TLS, de acordo com a IETF (Sheffer et al., 2015).

O protocolo Diffie-Hellman pode também ser generalizado a fim de abranger grupos de curvas elípticas, tema abordado na próxima secção.

2.2.4.2 Elliptic Curve Diffie-Hellman Key Exchange (ECDHKE)

Esta técnica recente, deriva da Criptografia de Curvas Elípticas (ECC), uma família de algoritmos de chave pública, que proporciona um nível de segurança próximo do RSA e dos sistemas DL, com a vantagem de utilizar operandos significativamente mais curtos (160–256 bits vs. 1024–3072 bits na versão convencional).

Em muitos dos casos, nomeadamente quando aplicada a chaves grandes, apresenta um melhor desempenho em relação ao RSA e a esquemas do tipo DL. A diferença de desempenho é visível ao nível da largura de banda, devido ao uso de chaves e/ou assinaturas mais curtas, mas é sobretudo evidente em termos computacionais, consequência do facto de serem executados maioritariamente cálculos relacionados com operações algébricas em lugar de operações de aritmética modular (Paar and Pelzl, 2010).

Uma vez que a ECC é baseada no problema do logaritmo discreto, significa que pode ser aplicada a protocolos do tipo DL tais como o Diffie-Hellman, surgindo assim o Elliptic Curve

Diffie-Hellman Key Exchange. Esta técnica difere do DHKE convencional principalmente no protocolo inicial de configuração, no qual são definidos os parâmetros de domínio, da seguinte forma:

1. Escolha de um primo p e de uma curva elíptica (E), definida pela equação:

$$(a) E : y^2 \equiv x^3 + a \cdot x + b \pmod{p}.$$

2. Escolha de um elemento primitivo, $P = (x_p, y_p)$

Os parâmetros de domínio são compostos pelo primo p , coeficientes a e b que definem a curva e pelo elemento primitivo P , também designado de ponto base.

A geração de uma curva elíptica válida é um processo difícil, dado que para ser considerada uma curva segura implica que esta demonstre uma série de propriedades estabelecidas como padrão (Paar and Pelzl, 2010). Por essa razão, em termos práticos, na aplicação de ECC é usado um conjunto de curvas elípticas pré-definidas, consideradas seguras, que normalmente corresponde à lista padrão de 15 curvas recomendada pelo NIST (Kerry and Gallagher, 2013).

O protocolo principal do ECDHKE é similar ao do DHKE convencional no que diz respeito ao fluxo de comunicação, diferindo apenas nos parâmetros transmitidos e nas operações que cada uma das partes tem de realizar. O diagrama ilustrado na figura 2.13 apresenta as operações e trocas de informação entre duas partes (Alice e Bob).

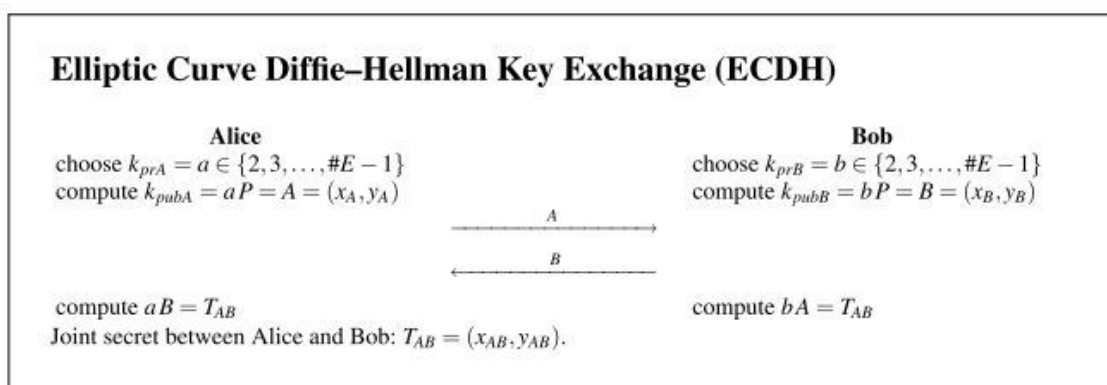


Figura 2.13: Elliptic Curve Diffie-Hellman Key Exchange (Paar and Pelzl, 2010).

Em resumo os passos necessários para obter a chave secreta comum, partindo da mesma curva e ponto inicial, são:

1. A Alice e o Bob escolhem dois inteiros grandes a e b representando a chave privada respectiva;
2. São geradas as chaves públicas A e B através da multiplicação da respectiva chave privada pelo ponto base;

3. Ambas as partes transmitem a sua chave pública ao parceiro;
4. A chave secreta é o resultado das seguintes operações:
 - (a) A Alice multiplica a sua chave privada pela chave pública do Bob: $T_{AB} = aB$
 - (b) Analogamente Bob calcula: $T_{AB} = bA$

Uma vez que:

- A Alice calcula $aB = a(bP)$ e o Bob calcula $bA = b(aP)$.
- E dado que a multiplicação é associativa, ambos chegam ao mesmo resultado, i.e., o ponto $T_{AB} = abP$.

Qualquer um dos valores da coordenada pode ser usado como entrada de um algoritmo de encriptação AES. No entanto apenas deverá ser usado um deles, uma vez que não são independentes e facilmente pode ser obtido um a partir do outro, em termos práticos é normalmente usado o valor da abcissa (Paar and Pelzl, 2010).

Relativamente a considerações de segurança do protocolo, durante a sua execução são apenas revelados os parâmetros a, b, p, P, A e B e para um atacante conseguir descobrir a chave secreta partilhada, teria de calcular $T_{ab} = a \cdot b \cdot P$, comumente chamado de *Elliptic Curve Diffie–Hellman problem* (ECDHP), sendo necessário resolver um dos seguintes problemas de logaritmos discretos:

$$a = \log_p A \quad \text{ou} \quad b = \log_p B$$

Se a curva elíptica usada for considerada segura criptograficamente, e.g., uma das curvas da lista de curvas padrão referida anteriormente, os ataques conhecidos contra *Elliptic Curve Discrete Logarithm Problem* (ECDLP) são mais fracos do que os melhores algoritmos conhecidos para resolver DLP e do que os melhores algoritmos de fatorização (usados para ataques RSA). De qualquer forma, para garantir um nível alto de segurança a longo prazo, para além da escolha de uma curva segura é recomendada a utilização de um primo p com 256 bits, que resulta num nível de segurança de 128 bits, i.e., uma complexidade de ataque de 2^{128} (Paar and Pelzl, 2010).

2.3 Conclusão

Em termos de mecanismos de transferências de chaves, uso de qualquer uma das técnicas está dependente da garantia de escolha de parâmetros de domínio seguros. A tendência aponta para que o *ephemeral*-ECDHKE seja cada vez mais utilizado, dado que traz maiores vantagens em termos de desempenho relativamente à versão convencional.

Relativamente à escolha do tamanho das chaves usados nos vários algoritmos, as recomendações atuais da ENISA apresentam-se na figura 2.14. A escolha do tamanho das chaves deve basear-se nas recomendações padrão existentes numa ótica de curto/longo prazo, podendo ser adaptados de acordo com as especificações do protocolo ou sistema implementado. Os valores referidos na figura pressupõem que não existem avanços revolucionários ao nível da computação quântica.

	Parameter	Legacy	Future System Use	
			Near Term	Long Term
Symmetric Key Size	k	80	128	256
Hash Function Output Size	m	160	256	512
MAC Output Size	m	80	128	256*
RSA Problem	$\ell(n) \geq$	1024	3072	15360
Finite Field DLP	$\ell(p^n) \geq$	1024	3072	15360
	$\ell(p), \ell(q) \geq$	160	256	512
ECDLP	$\ell(q) \geq$	160	256	512
Pairing	$\ell(p^{k \cdot n}) \geq$	1024	3072	15360
	$\ell(p), \ell(q) \geq$	160	256	512

Figura 2.14: Tamanho das chaves recomendado pela ENISA na aplicação de vários algoritmos, * significa que o valor pode ser inferior dependendo da implementação ([Weintraub and Blunden, Bill, 2013](#)).

Capítulo 3

Proposta de solução

Analisando com maior detalhe o problema contextualizado no capítulo 1, encontramos vários aspetos que têm de ser considerados relativamente à questão da segurança (*security vs safety*). Na língua portuguesa, o conceito de segurança é ambíguo e engloba os dois termos, no entanto em sentido prático a segurança pode entender-se como o conjunto de métodos aplicado na proteção (i.e., *safety*) de uma pessoa ou um edifício por exemplo. Neste caso, o foco da segurança do protótipo deverá também estar na proteção daqueles que interagem com o veículo dado que o sistema está conectado ao funcionamento do mesmo.

A existência de interferência externa por terceiros não autenticados representa uma falha de segurança que pode levar ao mau funcionamento intencional do dispositivo. Esse mau funcionamento também pode resultar em danos materiais do veículo e / ou de integridade física de quem se encontrar no seu interior, bem como a de terceiros. Neste sentido é necessário criar mecanismos que permitam dotar o sistema da robustez fundamental para resistir a possíveis ataques.

Ao longo deste capítulo são retratadas as alternativas estudadas, bem como a especificação da solução selecionada e respetiva arquitetura. No capítulo 4 é abordada a fase de desenvolvimento relativa à implementação dos vários segmentos da arquitetura.

3.1 Alternativas estudadas

Nesta secção apresenta-se a lista das alternativas analisadas, evidenciada na tabela 3.1, orientadas às principais características da proposta de solução, ou seja, em relação às tecnologias usadas na *back end*, modelo de comunicação, sistema de integridade, confidencialidade e autenticação dos dados.

Após análise e discussão das várias hipóteses chegou-se à conclusão de que, das alternativas estudadas, seria melhor optar pela alternativa #5. A seguir são apresentadas as motivações desta escolha em detrimento das outras opções.

Proposta de solução

#	Back-end	Comunicação	Confidencialidade	Integridade	Autenticidade
1	Node.js	TCP	<i>Symmetric key</i>	<i>Hash chain</i>	Assinatura Digital
2		TCP	<i>Asymmetric key</i>	<i>Hash chain</i>	Assinatura Digital
3		TCP	<i>Asymmetric key</i>	<i>Image Single Hash</i>	Assinatura Digital
4		TCP	<i>Symmetric key</i>	HMAC	
5		TCP	<i>Symmetric key</i>	Assinatura Digital	
6		HTTPS			
7	<i>Third-party software</i>				

Tabela 3.1: Lista das principais características das alternativas analisadas

O uso da tecnologia Node.js justifica-se pela compatibilidade com o sistema existente e por facilitar o desenvolvimento flexível de aplicações de rede escaláveis onde há necessidade de gestão de ligações concorrentes. Node.js engloba um ecossistema de inúmeras bibliotecas e módulos e conta com o apoio de uma comunidade vasta e em crescimento, criando assim as condições necessárias para o desenvolvimento profícuo deste projeto.

Atualmente a comunicação com o dispositivo é realizada via TCP, descartando a hipótese de uso de um protocolo, e.g., HTTPS. Este protocolo introduziria *overhead* e pacotes de dados desnecessários no que diz respeito aos interesses da empresa, uma vez que a manutenção de uma ligação mais pesada leva a um maior consumo de largura de banda que é um fator custoso numa perspetiva de escala.

O uso de *third-party software* foi descartado e apenas considerado para efeitos de análise das plataformas existentes no mercado, isto porque retiraria o propósito do desenvolvimento de uma solução flexível e controlada que integre com o sistema existente e não implique a dependência futura de terceiros.

A aplicação de assinaturas digitais garante à partida a possibilidade de verificar a integridade e autenticidade dos dados, bem como o não-repúdio dos mesmos, daí ter sido a abordagem escolhida em relação a HMAC e *hash chaining*. A confidencialidade é garantida pela aplicação de encriptação simétrica, a chave simétrica utilizada é única e estabelecida no início de cada sessão.

3.2 Especificação da solução

A proposta de solução consiste numa plataforma de gestão de atualizações FOTA baseada numa arquitetura cliente-servidor, em que o cliente representa o microcontrolador, o qual recebe atualizações de *firmware* a partir do servidor. Assume-se a existência de um módulo do lado do cliente, compatível com o protocolo referido.

Numa perspetiva de alto nível, a plataforma aproxima-se do modelo apresentado na figura 3.1 e divide-se em dois componentes lógicos principais:

Front end - componente que expõe uma interface simples e intuitiva, que permite ao utilizador visualizar e carregar ficheiros de *firmware* para posterior distribuição bem como monitorizar

Proposta de solução

remotamente o progresso dos processos de transferência FOTA.

Back end - este componente encapsula dois módulos, um módulo responsável pela execução do protocolo de atualização de FOTA e um segundo módulo que funciona como uma API, para articulação com as funcionalidades implementadas no *front end*.

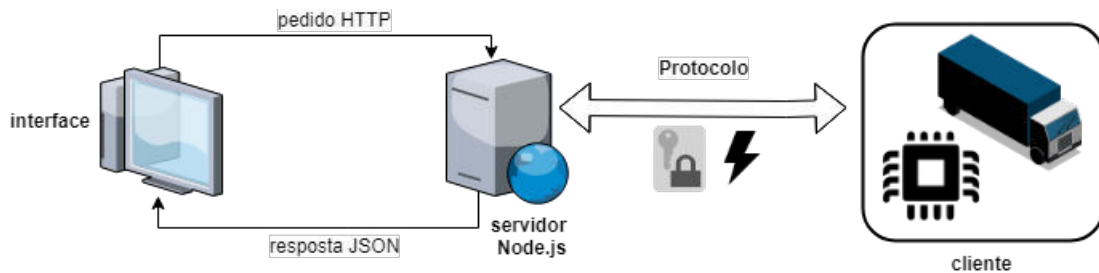


Figura 3.1: Abstração da proposta de solução.

A abordagem modular da proposta é um ponto importante para a sua futura evolução, tanto em termos de desenvolvimento como de escalabilidade, possibilitando a implementação e validação de cada um dos módulos de forma independente e em paralelo.

Esta proposta procura garantir elevadas características de segurança, ao nível da confidencialidade, integridade e autenticidade dos dados. Para tal são aplicados os seguintes conceitos:

- **Diffie-Hellman convencional** - com o objetivo de obter uma chave de sessão partilhada, comum ao servidor e dispositivo, para ser usada na encriptação simétrica a fim de mitigar possíveis ataques do tipo *man-in-the-middle* (MITM).
- **Assinatura digital** - com a finalidade de garantir o não-repúdio, a integridade e autenticação dos dados.
- **Encriptação simétrica** - usada a fim de manter a confidencialidade dos dados, comunicados por via de um canal inseguro.
- **Autenticação do dispositivo** - verificação do IMEI do dispositivo após conexão ao servidor.

Os tópicos referidos são empregues com o intuito de atribuir várias camadas de segurança à proposta de solução, a fim de minimizar vulnerabilidades, mitigar potenciais ataques e interferências de terceiros, conferindo uma maior robustez ao protocolo e a toda a plataforma.

O diagrama apresentado na figura 3.2 representa as várias etapas relativas ao ciclo de vida de uma atualização FOTA. Atualmente o dispositivo envia dados ao servidor com uma cadência de $\approx 1s$, o que significa que com a integração do protocolo de atualização FOTA, o novo cenário de funcionamento passa pelo dispositivo comunicar, com uma determinada frequência (diária,



Figura 3.2: Ciclo de vida de uma atualização FOTA.

semanal etc), a sua versão de *firmware* e em caso de existência de atualização proceder de acordo com o protocolo de transferência.

A arquitetura e os casos de uso da plataforma são detalhados na próxima secção.

3.3 Arquitetura

Nesta secção é retratada a arquitetura relativa à proposta de solução detalhada anteriormente, com o objetivo de dar a conhecer ao leitor a estrutura e o funcionamento dos vários módulos que compõem a plataforma e também a forma como eles se relacionam. São apresentadas também considerações acerca das limitações e pressupostos da arquitetura apresentada.

3.3.1 Pressupostos

Previamente ao desenho da arquitetura da plataforma existiram algumas considerações que foram assumidas à partida:

- A infraestrutura que aloja o servidor é considerada segura e de alta disponibilidade;

- O dispositivo possui a versão de *firmware* mínima para executar o protocolo de atualização FOTA;
- O dispositivo tem acesso a uma infraestrutura de ligação à rede Internet;
- O dispositivo tem espaço de memória *flash* suficiente para armazenar uma nova versão de *firmware*;
- O dispositivo contém as configurações necessárias para se conectar à plataforma.
- O acesso físico ao dispositivo por terceiros pode comprometer a segurança do mesmo (acesso a chaves, código etc).

3.3.2 Arquitetura física

Numa perspetiva abrangente, o protótipo segue a arquitetura apresentada na figura 3.3, onde estão representadas as ligações entre os 3 principais componentes: Dispositivo, Utilizador e Plataforma.

A plataforma incorpora dois serviços concorrentes: o módulo que executa o protocolo e o módulo que suporta a aplicação *web*. Suporta também múltiplos acessos concorrentes aos seus serviços, tanto de dispositivos como de utilizadores.

O dispositivo interage com a plataforma conectando-se ao serviço que executa o protocolo, via TCP. O utilizador interage com a plataforma através de um *web browser*, acedendo à página da interface.

Na figura 3.3 os serviços que executam o protocolo e aplicação *web* encontram-se representados no mesmo nó, no entanto a arquitetura definida permite isolar o serviço que executa o protocolo, facilitando a sua replicação em vários nós e permitindo que no futuro a plataforma escale de acordo com o crescimento do número de dispositivos.

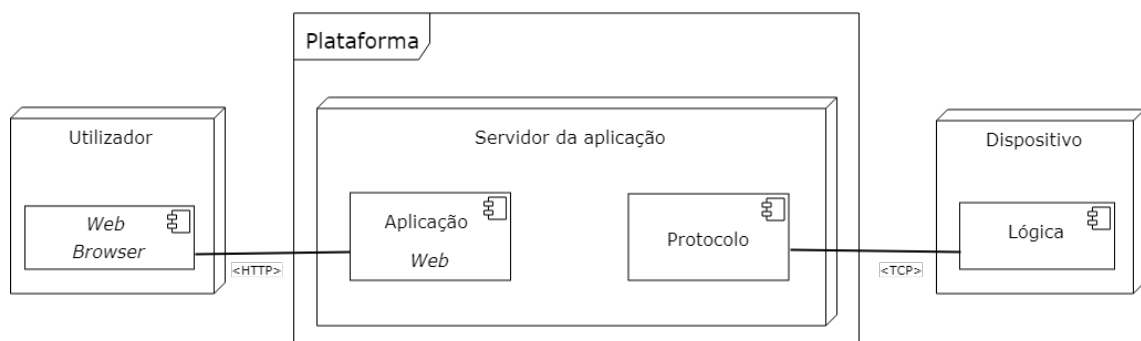


Figura 3.3: Diagrama da arquitetura física da plataforma.

3.3.3 Casos de uso

Os casos de uso associados à plataforma contam com vários atores, que constituem as diferentes entidades que podem interagir simultaneamente. A lista completa dos atores é apresentada de seguida:

- **Utilizador** - Utilizador que acede à plataforma através da interface *web*.
- **Interface *web*** - Aplicação *web* que fornece um conjunto de funcionalidades para interagir com a plataforma;
- **Servidor** - Servidor onde assenta toda a lógica associada ao funcionamento do protocolo de atualização FOTA e da interface *web*;
- **AddVolt API** - API para acesso a funcionalidades da plataforma da AddVolt (autenticação, carregamento de ficheiros etc);
- **AWS S3 API** - API para acesso a funcionalidades da Amazon Web Services S3 (serviço de armazenamento de dados na *cloud*);
- **Dispositivo** - microcontrolador alvo de atualização de *firmware*.

O caso de uso generalizado corresponde a uma atualização de *firmware*, que segue o ciclo de vida demonstrando anteriormente na figura 3.2. Este caso de uso pode ainda ser subdividido em dois casos de uso diferentes: carregamento de um ficheiro de *firmware* e execução do protocolo FOTA. Ambos os casos podem ser divididos em várias etapas, de acordo com a sequência ilustrada na figura 3.5.

No caso do carregamento de um ficheiro de *firmware*, destacam-se quatro etapas principais:

- Autenticação do utilizador na plataforma, através da interface *web*, relativa à secção "Autenticação" da figura 3.5;
- Carregamento do ficheiro na página respetiva, como se mostra na secção "Carregamento" da figura 3.5;
- Aprovação do lançamento de uma nova atualização, correspondente à secção "Lançamento" da figura 3.5;
- Monitorização do progresso do protocolo, visível na secção "Monitorização" (secção 'loop' inclusive) da figura 3.5.

Durante a execução do protocolo FOTA, cuja sequência de ações é exposta na figura 3.6, distinguem-se três etapas, algumas delas apenas são executadas após verificar-se o cumprimento

de algumas condições (i.e., autenticação inicial bem sucedida e versão de *firmware* do dispositivo desatualizada):

- Aprovação inicial do dispositivo (autenticação e versão de *firmware*), como se mostra na secção "Aprovação" da figura 3.6;
- Estabelecimento de uma chave de sessão, ilustrada na secção "Diffie-Hellman" da figura 3.6;
- Transferência do ficheiro de *firmware*, relativa à secção "Transferência" da figura 3.6.

A implementação de cada uma destas fases e correspondentes etapas é descrita com maior detalhe no próximo capítulo.

3.3.4 Dispositivo e *Firmware*

Ao longo do documento tem-se feito referência ao dispositivo de uma forma abstrata. Nesta secção é descrita a sua arquitetura e relações bem como o processo de instalação de *firmware* em linhas gerais.

O "Dispositivo" que comunica com o servidor a fim de receber atualizações FOTA é demonstrado na figura 3.4 com a legenda MLU (*Management Logging Unit*). A MLU consiste num nó de *hardware* contendo um microcontrolador (*chip*), com a particularidade de poder estabelecer uma ligação à rede Internet via GSM, tornando assim possível a comunicação de dados *over-the-air*.

A MLU encontra-se ligada a outros nós através da linha CAN (*CAN bus*), que consiste num sistema físico de comunicações usado na indústria, maioritariamente em veículos. Tal como referido na descrição da MLU, cada nó possui um microcontrolador. Este *hardware* integra também uma memória *flash* onde está armazenado o *firmware* responsável pelo funcionamento do microcontrolador.

No âmbito desta dissertação considera-se apenas a interação com o nó da MLU, responsável pela execução do protocolo de atualização FOTA. A instalação do *firmware* recebido pela MLU no respetivo nó é abstraído, não havendo qualquer controlo do processo a esse nível. O PC surge no diagrama com intuito de demonstrar que o *firmware* pode ser também transferido acedendo diretamente à linha CAN.

O *firmware* transferido sofre um *parsing* do lado do servidor a fim de remover alguns dados redundantes, reduzindo alguma sobrecarga ao nível do volume de dados transmitidos, que é um fator importante do ponto de vista da escala da plataforma.

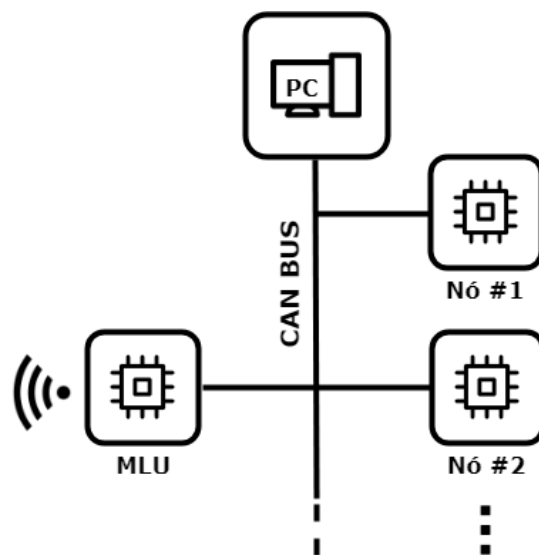


Figura 3.4: Arquitetura do dispositivo e respectivas ligações.

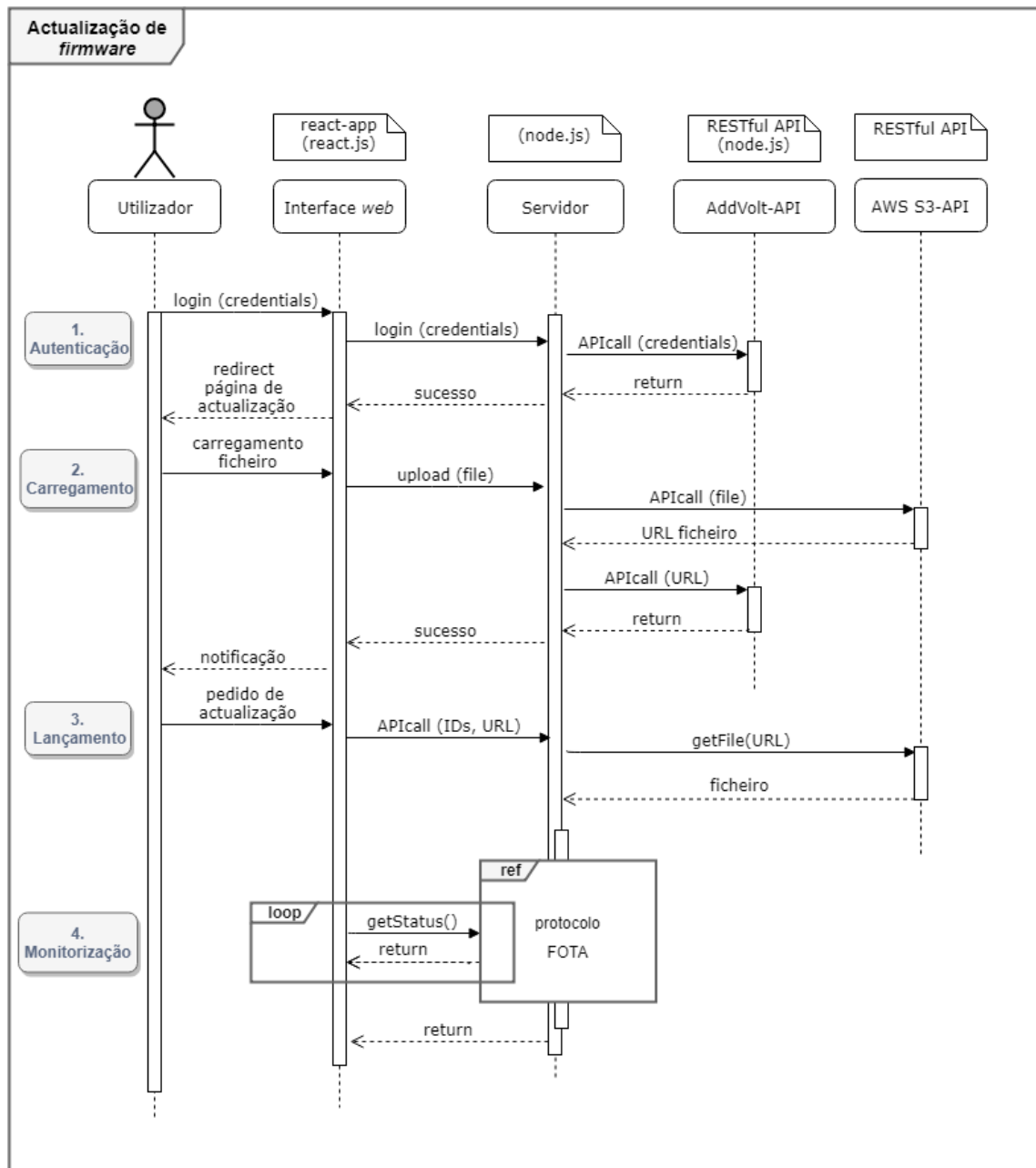


Figura 3.5: Diagrama de sequência referente ao caso de uso de atualização FOTA.

Proposta de solução

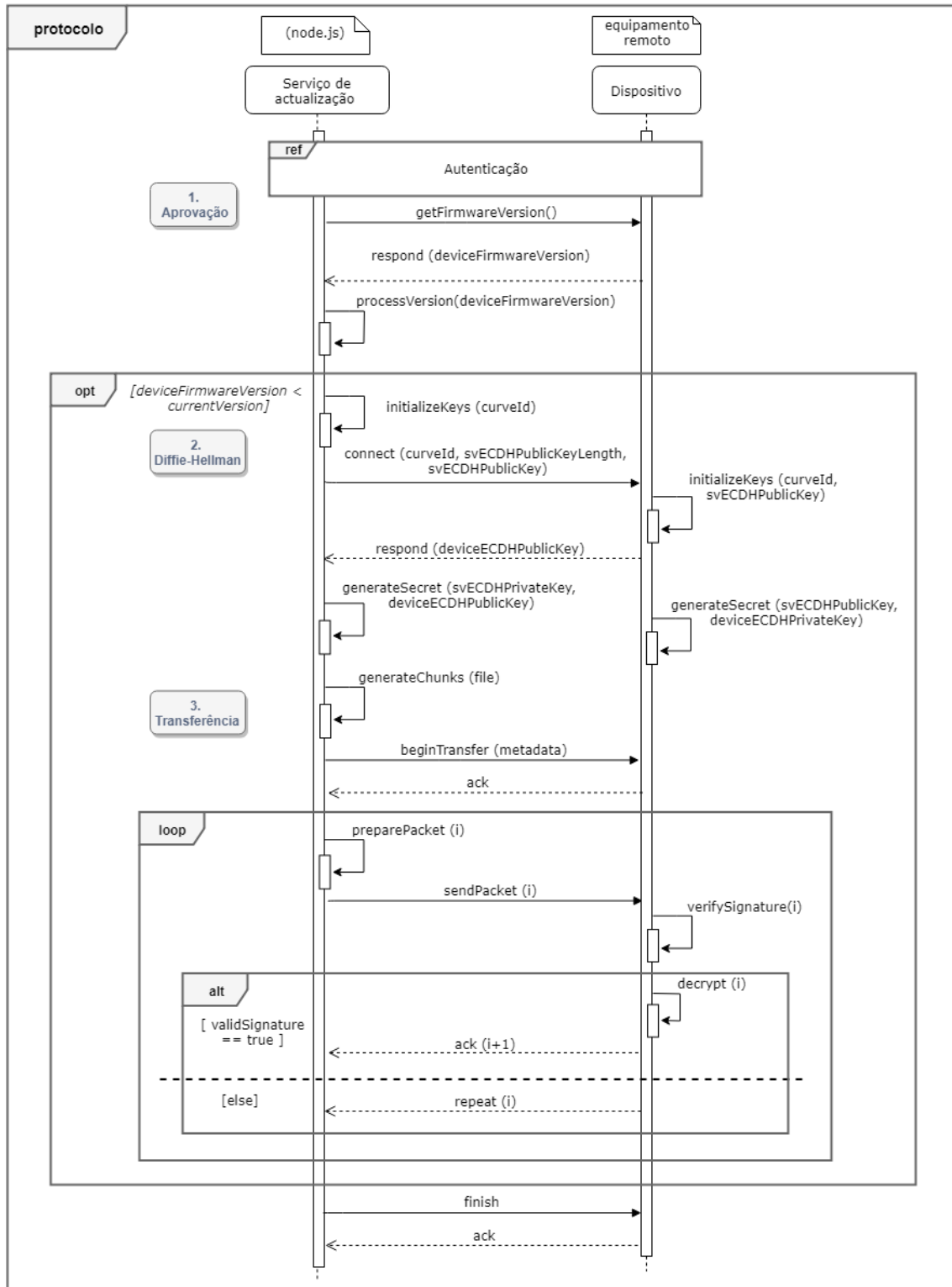


Figura 3.6: Diagrama de sequência relativo à execução do protocolo.

Capítulo 4

Implementação

Neste capítulo são abordadas as tecnologias usadas na implementação da plataforma e a forma como elas interagem entre si na estrutura da aplicação. É também descrita a estrutura do projeto, em termos de divisão dos ficheiros em módulos. É detalhado com maior pormenor o funcionamento dos servidores implementados e da interface *web* assim como a dinâmica do protocolo e implementação das funcionalidades de segurança.

4.1 Restrição do âmbito

Antes de iniciar a descrição da implementação é necessário ressaltar que foi necessário restringir o âmbito da arquitetura a fim de ter um protótipo finalizado no prazo estipulado.

No início do desenvolvimento chegou-se à conclusão de que seria inviável implementar, dentro do prazo previsto, todas as funcionalidades apresentadas no capítulo 3, devido à complexidade associada aos conceitos de criptografia propostos e integração com o dispositivo.

Como tal, decidiu-se manter o foco na implementação das partes centrais da proposta, ou seja, no desenvolvimento do protocolo de acordo com os requisitos de segurança (confidencialidade, integridade, autenticidade) e da interface *web* para monitorização da execução do protocolo. Foi consequentemente excluído do planeamento, o desenvolvimento da conexão a APIs externas (AWS-S3 API, AddVolt API) como ilustrado na figura 4.4. Optou-se por excluir a parte supracitada, por se ter entendido que continha relevo académico.

De qualquer forma, dada a natureza modular da plataforma e uma vez que essas funcionalidades foram consideradas no desenho da arquitetura, facilmente podem vir a ser integradas na implementação atual, numa fase futura.

Como nota adicional, após implementação da fase de *Diffie-Hellman* e por motivos de compatibilidade com a implementação do protocolo do lado do dispositivo, foi necessário alterar a implementação passando a ser utilizado o método ECDH, resultando em mudanças tanto ao nível da arquitetura como da lógica do protocolo.

4.2 Tecnologias

As tecnologias usadas para implementar a plataforma estão evidenciadas na figura 4.1.

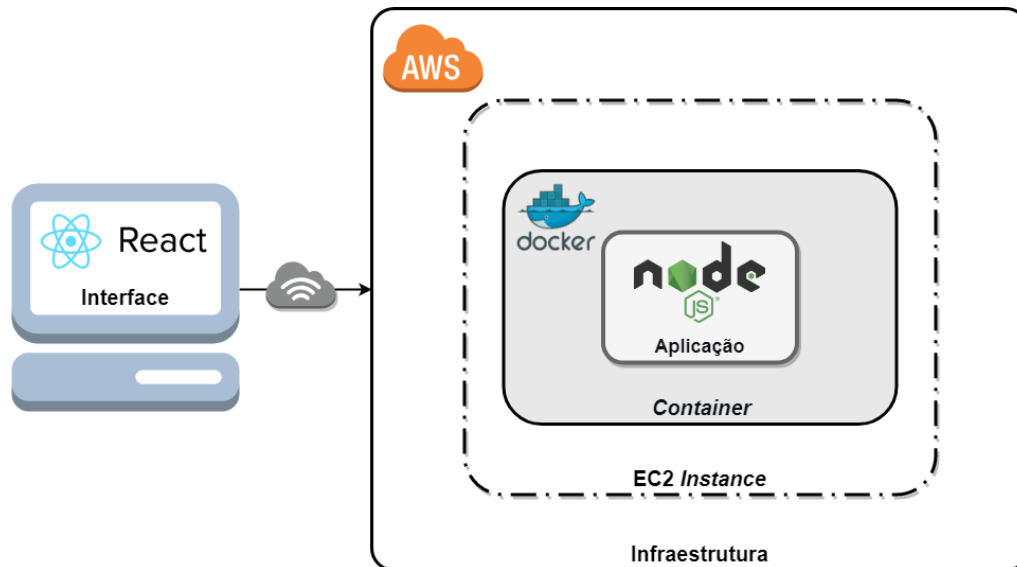


Figura 4.1: Diagrama de arquitetura e tecnologias usadas.

Para o alojamento do servidor foi usada a infraestrutura da AWS, nomeadamente o serviço Amazon Elastic Compute Cloud (Amazon EC2), onde foi escolhida uma máquina com especificações básicas, correspondendo à instância onde está a ser executada a plataforma. A escalabilidade desta infraestrutura é assegurada pela AWS, que permite lançar novas instâncias EC2 consoante a exigência em termos do volume de dispositivos e utilizadores.

Ao nível do *front-end*, para a implementação da interface, foi usada a *framework React*, numa perspetiva de facilitar uma integração futura na *dashboard* existente na AddVolt, a qual assenta na mesma tecnologia. *React* é uma *framework* popular na criação de UI's cujas principais inovações são a manipulação da interface através de Virtual DOM e o fluxo de dados *one-way* (Facebook, 2018).

Ao nível do *back-end*, tal como proposto, foi desenvolvido recorrendo à tecnologia Node.js (v8.10.0) (Foundation, 2018), utilizando funcionalidades da especificação do ECMAScript 6 (ECMA International, 2011). A fim de manter uma sintaxe de código padronizada e coerente, foi utilizado um *linter* (ESLint), ao longo de todo o desenvolvimento. Em termos de módulos externos destaca-se o uso das *frameworks Express*, *PM2* e *Mocha*, descritas mais à frente.

No que diz respeito à parte de desenvolvimento e operações (DevOps), para a "containerização" da aplicação foi utilizada a plataforma Docker (Inc, 2018), que permite encapsular uma aplicação num *container* virtual, abstraindo a aplicação do Sistema Operativo (SO) e permitindo

que vários *containers* sejam executados concorrentemente dentro da mesma instância. O facto de isolar a aplicação do Sistema Operativo (SO), permite que a aplicação seja executada tanto em ambientes Windows como Linux, tornando-a numa aplicação multi-plataforma.

Para o lançamento de novas instâncias, basta utilizar o *template* de configuração (Dockerfile) já criado, replicando-o para cada instância, facilitando assim a escalabilidade da plataforma.

Ao longo do desenvolvimento foi utilizada a tecnologia Git para controlo de versões e gestão de armazenamento do código ([git scm, 2018](#)). Todo o código desenvolvido encontra-se armazenado num repositório privado na *cloud*, alojado no serviço Bitbucket.

Com a finalidade de conseguir relançar automaticamente a aplicação em caso de falha, foi utilizado o módulo *PM2* que consiste num gestor de processos em ambiente de produção para Node.js. Esta ferramenta permite ainda obter métricas de desempenho do sistema (CPU, RAM etc) bem como alertas de falhas.

Para o desenvolvimento de testes unitários foi utilizado o módulo *Mocha* combinado com o módulo *Istanbul* a fim de avaliar a cobertura de código resultante da execução dos testes unitários.

4.3 Estrutura do projeto

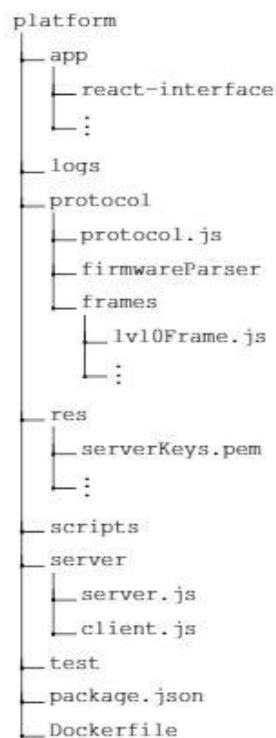


Figura 4.2: Estrutura do projeto em árvore.

O projeto encontra-se dividido em módulos, seguindo a estrutura apresentada no diagrama 4.2: módulo *app* que contém toda a lógica relativa à interface *web*; módulo *protocol* que inclui a lógica

Implementação

de manipulação de tramas, *parsing* de *firmware* e lógica principal de funcionamento do protocolo; módulo *server* que contém a lógica do servidor e do cliente; módulo *utils* que contém uma série de métodos utilitários; módulo *test* onde se encontram todos os teste unitários implementados.

Existem ainda pastas relevantes: pasta *res* armazena todos os recursos, ficheiros carregados a partir da interface *web*, chaves RSA, ficheiro de *firmware*; pasta *logs* que contém os ficheiros de registo internos; pasta *scripts* que contem *scripts* de automatização de tarefas.

Destacam-se ainda os ficheiros: *package.json* onde se encontram definidas as dependências do projeto e os scripts de execução; *Dockerfile* que contém as configurações para executar a aplicação dentro de um Docker *container*.

4.4 Servidor(es)

A plataforma é executada como um processo único que contém dois servidores, um servidor HTTP assente na *framework* Express relativo à interface *Web* e um servidor TCP assente na biblioteca Net (nativa) relativo à execução do protocolo, que por defeito estão à escuta nas portas 15000 e 16000 respetivamente.

O servidor HTTP é responsável pelo encaminhamento dos pedidos recebidos a partir da interface bem como as respetivas respostas, interagindo com o servidor TCP através dos seus métodos públicos:

```
protocolServer.getConnections() - recebe a lista de conexoes  
    armazenadas no servidor TCP, posteriormente filtradas.  
protocolServer.uploadFile(filename, data) - armazena um ficheiro de  
    firmware no sistema de ficheiros do servidor TCP, na pasta 'res/  
    firmware/'  
protocolServer.getFirmwareUploadsList() - recebe uma lista com o  
    nome dos ficheiros de firmware armazenados no servidor TCP
```

O servidor TCP é responsável por lidar com as conexões dos dispositivos e executar toda a lógica associada ao protocolo FOTA bem como os métodos de suporte à interface, referidos anteriormente. Durante a sua execução, o servidor TCP mantém uma lista de dispositivos conectados, cada conexão é apenas adicionada à lista se o dispositivo receber uma atualização de *firmware*.

Para cada conexão, o servidor aguarda de forma assíncrona pelo envio de uma mensagem, cada mensagem recebida é convertida numa trama com um formato específico e passada ao método raiz. Este método é responsável por encaminhar cada trama para o respetivo *handler* de acordo com o seu tipo, se for desconhecido a trama é descartada. A estrutura das tramas é analisada em maior detalhe na próxima secção.

relacionadas com a manutenção dos requisitos de segurança, integridade e autenticidade. No final da secção é especificada a dinâmica do protocolo, a fim de dar uma visão em profundidade do funcionamento do protocolo FOTA.

4.5.1 Estrutura das tramas

As tramas trocadas entre o cliente e o servidor TCP seguem uma estrutura por níveis tal como é apresentado na figura 4.3. A descrição de cada nível e respetivos campos é expressa na figura 4.3:

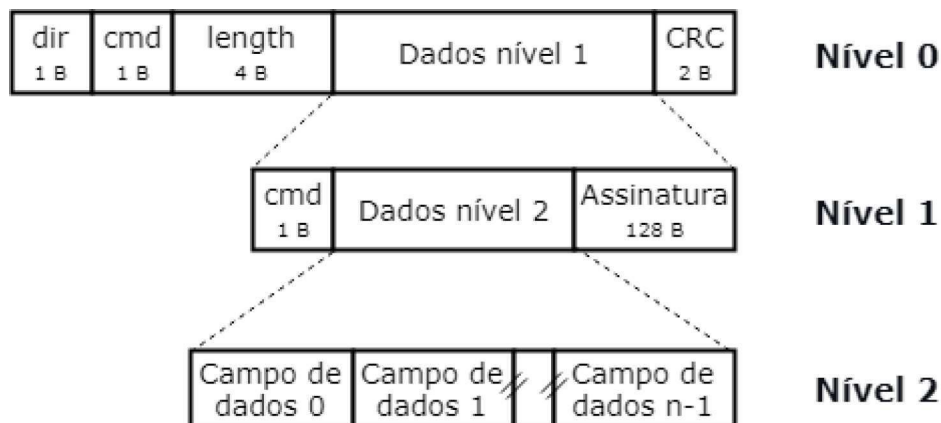


Figura 4.3: Representação da estrutura da trama por níveis.

- Nível 0
 - *dir* - representa o sentido da trama, cliente-servidor ou vice-versa;
 - *cmd* - especifica o tipo de trama, neste âmbito apenas são abordadas as tramas do tipo "Firmware update";
 - *length* - comprimento dos dados de nível 1, expresso em *bytes*;
 - *CRC* - *Cyclic Redundancy Check*, *checksum* relativo aos dados da trama a fim de garantir um grau mínimo de integridade, funcionando como uma camada inicial de verificação de integridade de dados. O sistema de comunicação dispositivo/servidor atualmente usada pela empresa faz uso do CRC, por essa razão implementou-se também neste protótipo tendo em vista facilitar a integração desse sistema nesta plataforma.
- Nível 1 - Os campos deste nível variam conforme o valor do parâmetro "cmd" de nível 0.
 - *cmd* - Especifica o subtipo de trama (identificação de diferentes erros, troca de metadados e chaves, transferência de um segmento de *firmware*, etc);
 - *assinatura* - assinatura digital relativa aos dados de nível 2.
- Nível 2 - Os campos variam dependendo do parâmetro *cmd* de nível 1.

Implementação

- *assinatura* - assinatura digital relativa aos dados de nível 2.
- Nível 2 - Os campos variam dependendo do parâmetro *cmd* de nível 1.

O processamento das tramas está dividido em três camadas conforme a estrutura referida, resultando numa separação lógica dos métodos de cada nível. Cada camada é constituída por um construtor, vários *parsers* e geradores de tramas. Os *parsers* de cada nível são responsáveis por manipular o segmento da trama (representada em hexadecimal) desse nível, separando os vários campos a fim de construir um objeto com as propriedades relativas aos campos desse nível.

Cada construtor chama o seu respetivo *parser* e invoca o construtor do nível acima, o qual recebe parte da trama correspondente como parâmetro. No final é retornado um objeto com a estrutura mostrada no excerto 4.1:

```
frame { // lvl 0
  cmdData: ,
  CRC: ,
  dir: ,
  length: ,
  supLevelFrameLength: ,
  ... ,
  supLevelFrame : { // lvl 1
    cmdData: ,
    length: ,
    signature: ,
    ... ,
    supLevelFrame: { // lvl 2
      imei: ,
      ...
    }
  }
}
```

Excerto 4.1: Abstração de uma trama.

Neste caso apenas se verifica a construção de tramas até uma profundidade de nível 2, no entanto, no futuro pode evoluir para uma profundidade nível *n*, em que cada subtrama ('supLevelFrame') aponta para outra subtrama e assim sucessivamente até a referência ser nula, representando o último segmento de trama.

As tramas enviadas pelo servidor são construídas no módulo '*protocol*' por via do método genérico:

```
buildFrame(builder, builderOpts[, svPrivateKey])
```

Este método faz a articulação dos segmentos de trama dos vários níveis, juntando-os de forma a criar uma *string* correspondente à trama final que é depois retornada ao ciclo principal e enviada ao cliente. O método referido invoca o gerador (*'builder'*) recebido como parâmetro, responsável por gerar o segmento de trama de nível 2 com os dados recebidos no parâmetro *'builderOpts'*. Se o parâmetro *'svPrivateKey'* estiver presente, os dados de nível 2 são assinados e de seguida é invocado o gerador de trama de nível 1, por fim é invocado o gerador da trama de nível 0, onde é calculado o CRC e são preenchidos os restantes campos desse nível.

4.5.2 Segurança e criptografia

Na camada inicial de verificação de integridade, optou-se por usar a abordagem padrão CRC16-CCITT no cálculo do CRC, cuja implementação deriva do módulo externo *'crc-full'* que inclui um método *'compute()'* que retorna um CRC de 2 bytes. Este método é usado tanto na construção de uma nova trama a ser enviada pelo servidor como na validação de uma trama recebida de um cliente.

A validação do IMEI do cliente é realizada comparando-o com um valor exemplo pré-definido no servidor, abstraindo a ligação à base de dados de IMEIs existente na empresa. Qualquer valor recebido distinto do pré-definido resulta no lançamento de uma mensagem de erro.

A validação da versão de *firmware* é um processo análogo ao anterior, onde é comparado o valor de versão recebido com um valor pré-definido, *hardcoded* no servidor, se corresponder a uma versão igual ou superior, é retornada uma mensagem de erro.

A implementação das funcionalidades criptográficas tiveram por base o uso do módulo nativo *'Crypto'* na versão 10.5.0 .

Todas as mensagens são assinadas digitalmente usando chaves RSA pré-instaladas, tanto no servidor como no cliente. É gerado um par de chaves estipulando-se um tamanho de 128 bytes (1024 bits) para a chave privada.

O módulo mencionado anteriormente disponibiliza também uma API para configurar a assinatura digital (especificando por exemplo o algoritmo de *hashing* utilizado) e para assinar e verificar assinaturas digitais. Neste caso, devido ao tamanho da chave privada, são geradas assinaturas com um tamanho de 128 bytes.

No que diz respeito à implementação de *ephemeral*-ECDH, o servidor funciona como mestre, i.e., é responsável por definir os parâmetros de domínio, nomeadamente a curva elíptica utilizada. Neste caso foi definido o uso da curva *'secp256k1'* (Kerry and Gallagher, 2013), uma das curvas presentes na lista aprovada pelo NIST, a qual gera um segredo de 256 bits. A cada conexão é gerado um novo segredo, resultante da execução do e-ECDH, garantindo a característica de *Perfect Forward Secrecy*.

Cada segmento (*chunk*) de *firmware* é encriptado usando a implementação padrão AES e o modo de encriptação *Cipher Block Chaining* (CBC) associado a uma chave de 128 bits, cujo tamanho foi escolhido por motivos de desempenho. Esta chave corresponde ao segredo obtido

Implementação

no processo anterior, o qual é truncado para metade. O algoritmo de encriptação necessita de um vetor de inicialização de 128 bits. Este vetor é inicializado com valores pseudo-aleatórios. Este método gera um *array* de tamanho *size* bytes de dados pseudo-aleatórios. Para descriptar os dados é necessário recorrer ao mesmo vetor de inicialização usado no processo de encriptação.

4.5.3 Dinâmica do protocolo

O protocolo FOTA mantém para cada conexão um estado corrente do protocolo, que inclui também o estado da transferência. O estado do protocolo evolui de acordo com as máquinas de estado implementadas, tal como demonstrado no excerto de código seguinte:

```
class ProtocolState {
    constructor() {
        this.state = PROTOCOL_STATES.AUTH;
        this.transferState = null;
    }

    updateState() {
        switch (this.state) {
            case PROTOCOL_STATES.AUTH:
                this.state = PROTOCOL_STATES.VERSION;
                break;
            ...
        }
    }
}

class TransferState {
    constructor(numChunks, updtMetadata) {
        this.acknowledgedChunks = 0;
        this.numChunks = numChunks;
        this.repeatedChunks = 0;
        this.metadata = updtMetadata;
        this.state = TRANSFER_STATES.READY;
    }

    updateState() {
        switch (this.state) {
            case TRANSFER_STATES.READY:
                this.state = TRANSFER_STATES.TRANSFERRING;
                break;
            ...
        }
    }
}
```

Excerto 4.2: Representação parcial da abstração do estado do protocolo e máquinas de estados.

A implementação do protocolo segue um modelo de comunicação pedido / resposta, para cada pedido enviado pelo servidor deverá ser recebida uma resposta por parte do cliente, contudo não existe persistência das mensagens recebidas, sendo estas descartadas após serem processadas.

O facto do protocolo seguir o modelo de comunicação referido e mudar de estado a cada resposta, não podendo regressar a um estado anterior, mitiga ataques do tipo *replay*, uma vez que a

receção de uma mensagem duplicada, e.g., correspondente a uma fase anterior, resultaria numa mensagem de erro, visto que o algoritmo estaria à espera de uma resposta relativa à fase em que se encontra. As várias etapas encontram-se ilustradas na figura 3.6 do capítulo 3.

O protocolo inicia-se com a conexão de um novo dispositivo ao servidor e receção da trama de autenticação que inclui o IMEI do dispositivo. Após validação do IMEI o protocolo avança para a fase de processamento da versão. Caso a versão esteja atualizada ou possua um formato desconhecido, é enviado ao dispositivo uma trama de fim de transferência, indicando o término do protocolo. Caso contrário, se a versão estiver desatualizada, o protocolo avança para a fase de troca de chaves ECDH.

Na fase de troca de chaves, o servidor envia uma trama contendo o identificador da curva usada e a chave pública gerada. Em resposta ao pedido, o cliente envia uma trama contendo a sua chave pública, gerando de seguida o segredo. O servidor efetua o processo análogo, após receber a chave pública do dispositivo. De seguida avança para a fase de transferência de *firmware*.

Na etapa relativa à transferência de *firmware*, o servidor carrega para memória o ficheiro já na versão pós *parsing*, dividindo-o em partes (*chunks*). Cada *chunk* tem um tamanho máximo pré-definido de 10 KB, valor escolhido devido à especificação de memória interna do dispositivo. Previamente ao envio do primeiro segmento, é enviada uma trama contendo informações (metadados) da transferência: o número de segmentos a serem transferidos, o tamanho máximo de cada segmento, a versão de *firmware* e o nó alvo de atualização. Cada um dos segmentos é encriptado e encapsulado numa trama de nível 2 juntamente com o vetor de inicialização usado na sua encriptação.

Após receção bem sucedida, o cliente deverá enviar uma mensagem de confirmação indicando o identificador do pacote recebido, o qual é registado do lado do servidor. Se por algum motivo o cliente não conseguir validar a trama de dados recebida, deverá enviar uma trama de repetição de pacote. O servidor apenas reenvia um pacote se ainda não tiver recebido a sua confirmação impedindo assim o envio de pacotes anteriores.

Quando se verificar a confirmação da receção de todos os segmentos é enviada uma trama de fim de transferência, indicando a conclusão do protocolo. Após resposta do cliente é concluída a conexão, terminando o *socket* associado, finalizando assim a execução do protocolo para a conexão referida. Para efeitos de monitorização, é mantido um conjunto de metadados da conexão e informações sobre a transferência e atualização de *firmware* a fim de informar o utilizador através da interface *web*.

4.6 Interface

A interface comunica com o servidor através de pedidos HTTP ao servidor associado ao serviço e é composta por quatro páginas. A implementação parcial do encaminhamento dos pedidos

Implementação

é demonstrado no excerto 4.4. As páginas que compõem a interface *web* são descritas de seguida.

- Página de *upload* onde é possível carregar e listar os ficheiros de *firmware* armazenados no servidor.
 - Para carregar um ficheiro é submetido um pedido POST para a rota `/api/upload` contendo o ficheiro e respetivo nome, que em caso de sucesso passa a estar armazenado no servidor. O excerto 4.3 demonstra a implementação parcial deste pedido;
 - Para obter a lista de ficheiros de *firmware* armazenados é submetido um pedido GET vazio para a rota `/api/files` cuja resposta contém a lista de nomes dos ficheiros de *firmware* armazenados no servidor em formato JSON.
- Página de conexões onde é possível monitorizar o progresso do protocolo para cada cliente conectado.
 - O pedido GET à rota `/api/connections` recebe como resposta em formato JSON a lista de conexões (ativas e inativas) desde que o servidor associado ao serviço de execução do protocolo foi executado. Os metadados presentes na resposta contêm informações relativas ao estado da conexão, volume de dados transmitidos, tempos de transferência, etc.
- Página de logs, exhibe o conteúdo do ficheiro de *logs* interno à plataforma, permitindo ter uma visão mais baixo nível do estado do servidor, útil no rastreio de problemas e suporte a *debugging*.
 - O ficheiro de *logs* é recebido como resposta ao pedido GET à rota `/log`.
- Página de métricas, expõe o conteúdo do registo interno de métricas relativas a cada execução do protocolo FOTA.
 - O ficheiro de métricas é recebido como resposta ao pedido GET à rota `/metrics`.

```
sendFileDataToServer(filename, data) {  
  fetch(API_ENDPOINT,  
    {  
      method: 'POST',  
      body: JSON.stringify({ filename: filename, data: data }),  
      headers: new Headers({  
        'Content-Type': 'application/json'  
      })  
    })  
}
```

Excerto 4.3: Parte da função que submete o pedido POST para carregamento do ficheiro de *firmware*.

```
// protocol server log
router.get('/log', logHandler.readLog);

// protocol server metrics
router.get('/metrics', logHandler.readMetrics);

// active connections
router.get('/api/connections', connHandler.readConnections);

// upload file
router.post('/api/upload', uploadHandler.uploadFile);

//get uploaded files
router.get('/api/upload', uploadHandler.getUploadedFiles);
```

Excerto 4.4: Roteamento de pedidos HTTP relativos à interface *web*.

4.7 Limitações

A plataforma possui algumas limitações em termos de implementação, designadamente ao nível dos algoritmos criptográficos. Esta é uma limitação residual e prende-se com a validade do tamanho das chaves usadas nos algoritmos aplicados, que de acordo com as previsões do NIST referidas em 2.2, serão válidas até 2040. Esta data oferece ao sistema um intervalo de tempo bastante confortável para se adaptar a novos potenciais avanços na criptografia e computação quântica. A atualização do tamanho das chaves implica uma mudança ao nível da estrutura das tramas e da lógica do protocolo.

Existe também uma limitação em termos de desempenho na verificação das assinaturas digitais, a verificação via RSA é pouco eficiente, principalmente em dispositivos com recursos mais limitados.

O protocolo de comunicação está limitado à implementação assente em TCP apenas, a atualização para um protocolo como por exemplo HTTP, MQTT, etc, implica uma reestruturação em termos de arquitetura da plataforma.

O número de conexões concorrentes encontra-se limitado ao número de *sockets* disponíveis na máquina que executa o protocolo.

Ao nível da interface, a implementação atual assenta em no protocolo HTTP vulnerável a ataques MITM. Este é um dos aspetos considerados nos melhoramentos ao protótipo, descritos no capítulo 6.

Implementação

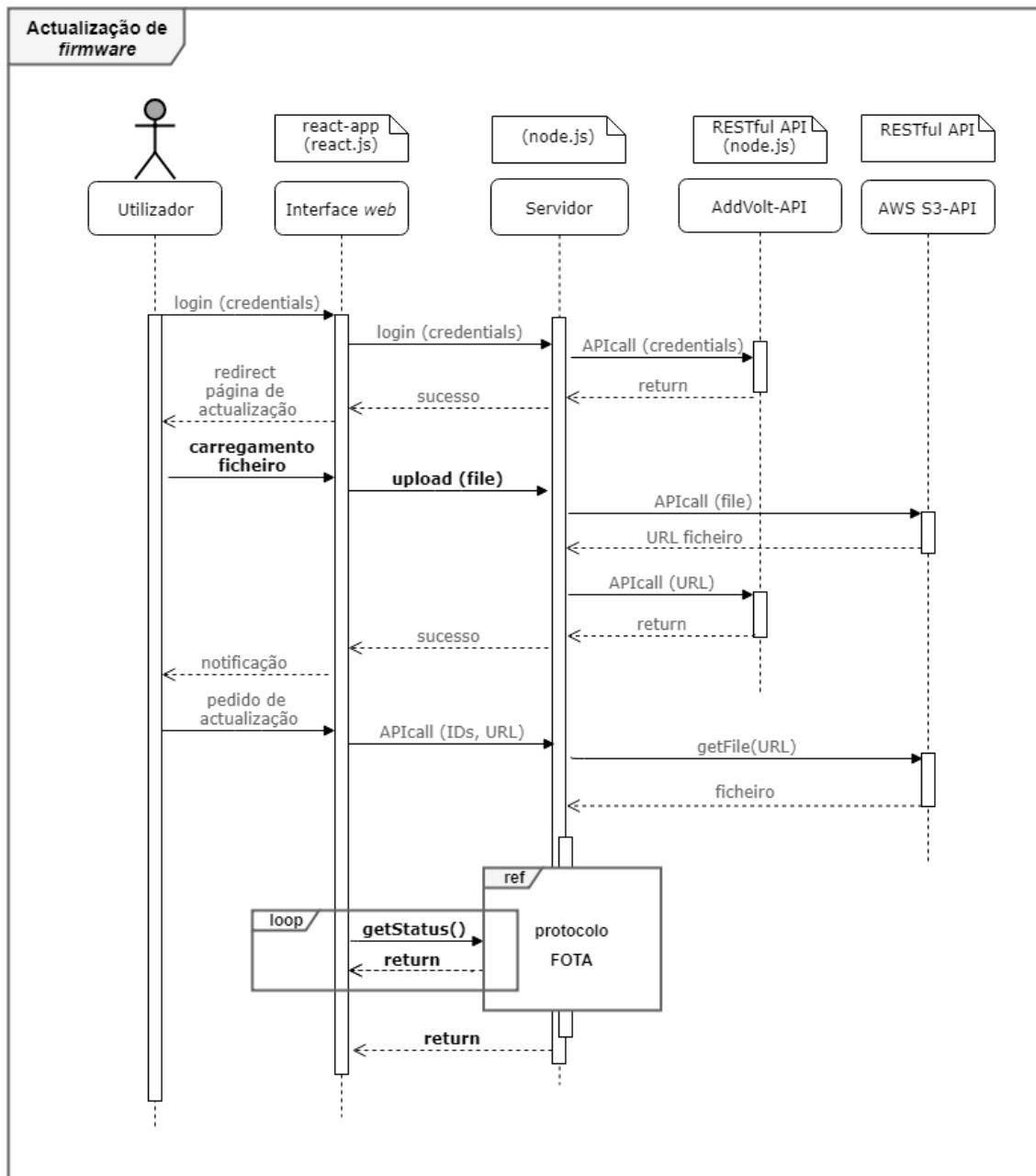


Figura 4.4: Diagrama de sequência do caso de uso de atualização FOTA após restrição de âmbito.

Capítulo 5

Resultados

Este capítulo aborda inicialmente as instruções de operacionalização da plataforma em ambiente de desenvolvimento e produção. De seguida é feita uma descrição da interface *web* implementada bem como das páginas associadas à mesma. Posteriormente são apresentados as métricas recolhidas e os testes realizados à plataforma. O capítulo é concluído com uma análise e discussão dos resultados obtidos.

5.1 Operação e execução

Para a execução local da plataforma é necessário ter instalada a versão 8.10.0 de Node.js e encontrando-se no diretório raiz do projeto, a partir da linha de comandos executar os comandos:

- `npm install`
- `npm run dev`

O primeiro comando instala as dependências necessárias e o segundo executa a aplicação. Para correr a plataforma dentro de um Docker *container* em ambiente de produção é necessário ter instalada a versão 18.03.1-ce de Docker e encontrando-se na raiz do projeto, a partir da linha de comandos executar o *script* "runDocker.sh" com o comando:

- `sh scripts/runDocker.sh`

Este *script* engloba os comandos relativos à criação de uma nova imagem a partir de um repositório remoto, adiciona o código da aplicação e inicia um *container* expondo as portas necessárias para o funcionamento da plataforma.

A interface *web* da plataforma estará disponível acedendo localmente a "localhost:3000". Em produção estará acessível a partir do endereço de IP público da máquina onde a plataforma está a ser executada. Para simular uma atualização FOTA pode-se correr num cliente local, para tal, encontrando-se na raiz do projeto, basta executar o comando:

- `node server/client.js`

5.2 Interface web

Nesta secção são apresentadas ilustrações da aparência da interface *web* e respectivas funcionalidades. Como documentado anteriormente, a interface é composta por duas páginas principais ("Connections" e "Upload") e duas páginas auxiliares ("Log" e "Metrics") direccionadas à equipa de engenharia.

5.2.1 Página de conexões

A página de conexões apresenta a lista de conexões em curso e terminadas, relativas à execução do protocolo FOTA. Na demonstração ilustrada na figura 5.1, são apresentados três estados possíveis de uma conexão:

- Execução do protocolo FOTA terminada;
- Transferência em progresso;
- Transferência cancelada.

Connections

IMEI	Address	Start Time	End Time	Connection	Firmware	Protocol State	Transfer State
01a101a101a101a101a101a101a101	5.249.66.64:63432	2018-06-23T15:54:09.948Z	2018-06-23T15:54:56.056Z	"closed"	MLU-01	FINISHED	FINISHED
01a101a101a101a101a101a101a101	5.249.66.64:63496	2018-06-23T15:54:55.022Z		"connected"	MLU-01	TRANSFER	82%
01a101a101a101a101a101a101a101	5.249.66.64:63509	2018-06-23T15:55:04.216Z	2018-06-23T15:55:30.388Z	"closed"	MLU-01	ERROR	57%
01a101a101a101a101a101a101a101	5.249.66.64:63541	2018-06-23T15:55:24.731Z		"connected"	MLU-01	TRANSFER	17%

Figura 5.1: Página de conexões da interface *web*.

No exemplo associado à transferência cancelada, foi simulado desconetando o cliente de forma brusca a fim de representar um dispositivo que, e.g., perde a conexão à rede GSM, sofre uma falha de *hardware* ou *firmware*.

Em cada conexão é possível visualizar diversas informações: IMEI, endereço IP remoto e porta, tempos relativos ao início e término da conexão, estado da conexão, nó de *hardware* alvo de atualização, estado do protocolo e progresso da transferência de *firmware*. Estas informações são úteis para uma gestão mais abrangente e em tempo real das atualizações em curso, permitindo facilmente identificar problemas, potenciando assim uma intervenção mais rápida e eficaz por parte da equipa de engenharia.

5.2.2 Página de carregamento de ficheiros

Na página "Upload" é possível visualizar a lista de ficheiros de *firmware* armazenados no servidor e também carregar novos ficheiros largando na área destinada a esse efeito ou através do sistema de ficheiros local do utilizador, como demonstrado na figura 5.2.

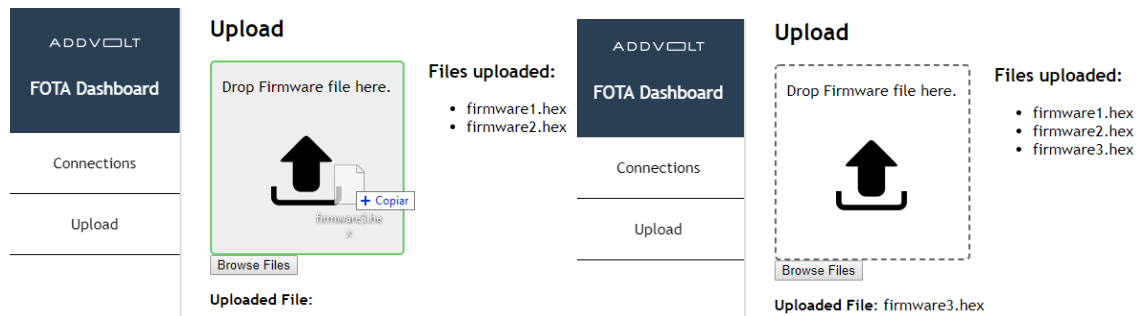


Figura 5.2: Carregamento de um ficheiro exemplo via *drag-and-drop* através da interface *web*.

5.2.3 Páginas auxiliares

A página de "log" serve apenas para expor o ficheiro de *log* interno ao servidor na *web*. Ao consultar esta página é possível ter uma visão de mais baixo nível dos detalhes de execução da plataforma desde o momento em que foi iniciada. Esta ferramenta torna-se especialmente útil em casos de *debugging*, facilitando o acesso ao *log* da plataforma sem recorrer a processos morosos, e.g., estabelecimento de uma conexão *ssh* à máquina. Na figura 5.3 é demonstrado uma parte do ficheiro de registo, onde é visível também a data e hora de cada operação, com as mais recentes surgindo primeiro.

```
==> Reading from logfile /platform/logs/protocol.log  
[Sort: recent files first]  
  
=====$=====
```

[23/06/2018 16:33:36]	[LOG]	Successfully uploaded file firmware3.hex
[23/06/2018 16:31:48]	[LOG]	Server started at: 0.0.0.0:15000
[23/06/2018 16:31:48]	[LOG]	Http process running on port 16000.
[23/06/2018 16:31:48]	[LOG]	

```
-----
```

Figura 5.3: Página de *log* da interface *web*.

A página de métricas apresenta, como ilustrada na figura 5.4 um série de métricas relativas a cada sessão de execução do protocolo, permitindo inferir dados como a duração da sessão, número de pacotes repetidos e o volume de tráfego de dados.

5.3 Métricas avaliadas

Com intuito de avaliar o desempenho da plataforma foram consideradas as seguintes métricas:

Resultados

```
=====
=====
=====
== | METRICS 56769 | ==
== Rec(B): 5647 ==
== Sent(B): 685801==
== repeated packets: 0 ==
== duration total (s): 46.11299999999983 ==
== duration handshake (s): 0.0500000000001819 ==
== duration transfer (s): 45.947999999999865 ==
=====
```

Figura 5.4: Página de métricas da interface *web*.

- Métricas do protocolo (por conexão)
 - Duração da sessão;
 - * Duração do *handshake* - Autenticação e verificação da versão;
 - * Duração da transferência.
 - Número de pacotes repetidos;
 - Volume de tráfego de dados.
- Métricas de desempenho da máquina (utilização de recursos)
 - CPU;
 - RAM.

Para recolher as métricas de desempenho da máquina foi utilizada uma ferramenta de monitorização denominada Keymetrics ([Keymetrics, 2017](#)). As métricas referidas na lista são analisadas na próxima secção, relativa aos testes realizados à plataforma.

5.4 Testes

Esta secção apresenta o conjunto de testes realizados à plataforma face às métricas referidas anteriormente. Os testes efetuados tiveram como objetivo demonstrar os requisitos especificados no capítulo 3 e dar uma visão gráfica do funcionamento da plataforma implementada, focando no seu comportamento ao nível do *back-end*.

Não foram realizados testes com um dispositivo (MLU) pelo facto de a implementação do lado do dispositivo não suportar a execução do protocolo no momento em que os testes foram efetuados. Não obstante, o dispositivo foi abstraído localmente e parametrizado de forma a aproximar-se da realidade.

As especificações das máquinas utilizadas nos ambientes de teste são as seguintes:

- Servidor (AWS) - 8 GB disco HDD, 1GB RAM, Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
- Cliente (local) - 250 GB disco SSD, 8 GB RAM, Intel(R) CORE(TM) i7-3537U CPU @ 2.00-2.50 GHz

Em cada teste definiu-se um tamanho máximo de 10 KB para cada *chunk*, onde o servidor armazena um par de chaves RSA próprio bem como a chave RSA pública do dispositivo e análogamente o cliente armazena um par de chaves RSA próprio e a chave RSA pública do servidor.

A lista de testes realizados apresenta-se de seguida, agrupada em três categorias: demonstração base da execução do protocolo FOTA, demonstração de requisitos (confidencialidade, integridade, autenticidade) e testes de escala.

5.4.1 Demonstração base do protocolo FOTA

Na figura 5.5 são evidenciados *logs* parciais das principais etapas do funcionamento do protocolo FOTA.

Inicialmente encontra-se ilustrada a nova conexão de um cliente e respetiva validação inicial. Na imagem (b) apresenta-se o registo da fase final de execução do Diffie-Hellman, onde é obtido um segredo comum (censurado na imagem), seguida pelo pré-processamento do ficheiro de *firmware* e envio de metadados ao cliente relativos à transferência. A imagem (c) reflete o envio de uma trama de confirmação de receção de um *chunk* (com id igual a 11) por parte do cliente e envio do pacote seguinte (com id igual a 12) por parte do servidor. Por fim na imagem (d) é demonstrada a fase final de execução do protocolo com a receção da confirmação da trama de fim de transferência e cessação da conexão com o cliente.

Ao longo deste capítulo, como é visível na imagem 5.5, existem várias informações censuradas com o intuito de proteger a confidencialidade do protocolo e impedir a aplicação de engenharia reversa ao mesmo.

5.4.2 Validação de requisitos

Nesta secção começa-se por demonstrar a funcionalidade de reenvio de um pacote, o cliente pode descartar um *chunk* por motivos de e.g integridade inválida, enviando como resposta uma trama de repetição em que é indicado o identificador do *chunk* a ser reenviado. O servidor apenas reenvia no caso de ainda não ter recebido anteriormente uma confirmação de receção do *chunk*.

Na figura 5.6 apresenta-se parte da trama relativa ao envio de um *chunk* na imagem (a) com alguns dados encriptados e assinatura. A imagem (b) corresponde ao pedido de reenvio. Apesar dos parâmetros do pacote serem os mesmos, o conteúdo final da trama varia em relação à trama enviada anteriormente uma vez que a encriptação gera valores diferentes a cada execução, que por sua vez origina valores distintos em termos de assinatura digital.

Resultados

```
0>40a0
[23/06/2018 17:44:18] [LOG] VALID SIGNATURE
[23/06/2018 17:44:18] [LOG] IMEI = 01a101a101a101a101a101a101
[23/06/2018 17:44:18] [LOG] Client [5.249.66.64:56769] sent:
.....
$.....
[23/06/2018 17:44:18] [LOG] New client connection from 5.249.66.64:56769
```

(a) Nova conexão

```
[23/06/2018 17:44:18] [LOG] Send metadata message.
[23/06/2018 17:44:18] [LOG] transfer pre processment done 35
[23/06/2018 17:44:18] [LOG] chunks length:35
[23/06/2018 17:44:18] [LOG] protocol: lines length 15441
[23/06/2018 17:44:18] [LOG] protocol: file length 694772
[23/06/2018 17:44:18] [LOG] starting file preprocessment
[23/06/2018 17:44:18] [LOG] Secret :
[23/06/2018 17:44:18] [LOG] RECEIVED DH CONNECTION REPLY FROM CLIENT
[23/06/2018 17:44:18] [LOG] Client [5.249.66.64:56769] sent:
```

(b) Diffie-Hellman e pré-processamento

```
[23/06/2018 17:44:32] [LOG] Chunk id 12 sent
[23/06/2018 17:44:32] [LOG] Client [5.249.66.64:56769] sent:
.....906
6e737e1fc.....
.....495f742c091.....
f280e5da
```

(c) Confirmação de receção e envio do *chunk* seguinte

```
[23/06/2018 17:45:04] [LOG] connection from 5.249.66.64:56769 closed
[23/06/2018 17:45:04] [LOG] connection from 5.249.66.64:56769 ended
[23/06/2018 17:45:04] [LOG] FW update transfer on client 01a101a101a101a101a101a101
finished!
[23/06/2018 17:45:04] [LOG] RECEIVED END of FW UPDT ack FROM CLIENT
```

(d) Finalização do protocolo

Figura 5.5: Registo das várias etapas da execução do protocolo FOTA.

De seguida é demonstrado um teste de validação do IMEI, neste caso de teste o cliente envia uma trama de autenticação com o valor de IMEI "293301a101a101a101a101a101a101", no entanto o servidor apenas considera válido o IMEI "01a101a101a101a101a101a101". A imagem (a) da figura 5.7 ilustra a trama enviada do lado do cliente, onde consta o valor de IMEI inválido referido. na imagem (b) ilustram-se as mensagens de erro resultantes da falha na autenticação do IMEI por parte do servidor.

De seguida, na figura 5.8 é demonstrado um caso de teste em que, após construção da trama do lado do cliente, são alterados três bytes para o valor 0xFF. Após processamento do lado do servidor verifica-se que o valor de CRC presente na trama não corresponde ao valor de CRC calculado, resultando na mensagem de erro exposta na figura.

Resultados

```
b1f164d66c962bc7973f1313e183c51b24c3fc52f84d869ce2d8e16d477bf6811bfdddcfc24b03879ff092115eeccf30712da47e2e4b3df6604560864a2931b6fc60e6912b0e9168961c029d7747dbd049b51cda906d62e74746507  
a918c9b6cf92f80a2f542a27798ae9475cd5d96bde52bcc7946235925fec736e99336965062a26f5300ad081f3c1aae2665d02e743af2c91ed7027dc1e5a8d02d03c6571e13540fb5381e7e9401cbac5745e0bd964823a1ea9270fe  
7f8f2705f06d538fb1fab5c2d093ec7a4ced62aef6bd486de2a6dd332e06fbc94c8a7473d4afbb5c02f62f95597e59a8c31acc8161cbe56f006f508108057babe70e78e50b1c33384f01b92cc6e8d92208dddc5ea78d19ad55  
372d813590a260ef5b2da3c3dfdfb178cf379dfb40cc06293d17405a1e77feeef1045ab8fffa8b3230e0e002764216ae419ab  
[23/06/2018 12:46:40] [LOG] Chunk id 32 sent
```

(a) Envio do *chunk* 32.

```
4d8d07f6067701586775e57487e1d59e4c2bd31f43215f3a6306d1a22ffb837e112e63677bbd10818712722abda7596ba1bde0d952a88bd05cfea7b6466286fbc63a78fe91dab41141d636e80e714fa4c06e60d40208dd8439eb1a  
e2f4eb2e25f7be10fa5dc4bfa59e46f7515935264a1abddc36c7c44f21fcc557423fcd4695babaed2ed978a1ed4ec221979ec9edce1753fb50b62600f0595d807c7bce0e98f73444722d01f4e53fb4ca316e2d235cececabfa27ba209  
d38a8c2d19a2c19ba89652bd99619f74e13bda6caf00cb1ef8aaf35726a8ee173a393636bb3fdae56b50eb3206a0653d987935141e5d810517af4e52fceb63a5e9aa1b12fc1090c2ff78757efaaafb16359dc23b5ee55d50a2fdd57d68  
3b015babd230023b6a7fb5b1084d7a9a18e21b8ce9a7a2cb66eb713491b89b9f0376b85862996bc3b9c6143b20a8064488  
[23/06/2018 12:46:41] [LOG] Chunk id 32 sent  
[23/06/2018 12:46:41] [LOG] Client sent Repeat Packet
```

(b) Pedido de reenvio do *chunk* 32

Figura 5.6: Reenvio de um pacote de dados

Na figura 5.9, analogamente ao caso de teste anterior, é demonstrado um teste no contexto de uma adulteração do campo de dados assinado, previamente ao cálculo do CRC. O segmento original era composto por dois bytes com o valor 0x00, posteriormente o primeiro byte é alterado e passa a ter o valor 0xFF. Uma vez que esta adulteração no campo de dados da trama acontece antes do cálculo do CRC, quando processada pelo servidor não vai levantar problemas em termos de integridade de CRC, no entanto no momento de validação da assinatura é lançado um erro, como esperado, indicando a presença de uma assinatura inválida.

Por último, foi realizado um caso de teste em que foram assinadas duas tramas iguais recorrendo a chaves RSA diferentes. A motivação de teste prende-se com a necessidade de prevenir casos em que é usada uma chave desconhecida para assinar uma dada trama. Como esperado, ao assinar com chaves diferentes, obtêm-se valores de assinatura distintos, tal como é ilustrado na figura 5.10.

O servidor recebe e processa a trama de autenticação, validando a assinatura digital com a chave pública correspondente ao IMEI presente na trama, a verificação é bem sucedida tal como é visível na imagem (a) da figura 5.10, confirmando-se a autenticidade do remetente. Na imagem (b) ilustra-se o processo de validação da assinatura análogo ao de (a), contudo neste caso a validação da assinatura não é bem sucedida, como esperado.

5.4.3 Testes de escala

Nesta secção são apresentados os testes de escala da plataforma e respetivos resultados. O objetivo deste grupo de testes foi avaliar o desempenho e capacidade de escala da plataforma, quando executada numa máquina com as especificações referidas anteriormente. Pretendeu-se também aferir a dinâmica na duração média das sessões e consumo de recursos com o aumento do número de conexões.

Para tal foram realizados testes com 1, 10, 50, 100 e 200 clientes conetados simultaneamente à plataforma. O primeiro caso de teste, com um único cliente conetado, serve como referência de comparação para os restantes casos de teste. Em cada um dos casos foi usado o mesmo ficheiro de *firmware*, com um tamanho de 679 338 bytes.

Os resultados obtidos relativos às métricas do protocolo encontram-se detalhados na tabela 5.1.

Resultados

```
Send auth msg:  
093301a101a101a101a101a101
```

(a) Trama com IMEI inválido enviada pelo cliente

```
[23/06/2018 10:02:08] [LOG] Error: IMEI is not valid!  
[23/06/2018 10:02:08] [LOG] Caught server socket error: Error: IMEI is not valid!  
[23/06/2018 10:02:08] [LOG] VALID SIGNATURE  
[23/06/2018 10:02:08] [LOG] IMEI = 293301a101a101a101a101a101a101  
[23/06/2018 10:02:08] [LOG] Client [5.249.66.64:54438] sent:  
#####293301a101a101a101a101a101a101#####  
#####  
#####
```

(b) Registo de falha na autenticação no servidor

Figura 5.7: Teste de validação da autenticação com IMEI desconhecido.

Na figura 5.11, demonstra-se a evolução na utilização de recursos da máquina ao longo da execução do protocolo em cada um dos casos de teste, apresentando para cada caso o pico máximo de consumo registrado. O resumo dos dados presentes nas imagens da figura é apresentado na tabela 5.2, representando uma aproximação dos valores máximos de consumo de recursos ao nível do CPU e RAM.

5.5 Discussão dos resultados

Nesta secção é realizada uma análise dos resultados obtidos, descritos na secção anterior.

Relativamente ao desempenho do protocolo FOTA, considerando valores de latência constante, consegue perceber-se através da figura 5.12 que independentemente do número de clientes conetados em simultâneo, aproximadamente 90% do tempo de execução do protocolo é dedicado à transferência de *firmware*, sendo o restante tempo usado na fase inicial de autenticação e na fase seguinte de troca de chaves via ECDH. Destaca-se também que a duração mais longa registada na execução do protocolo, em condições ótimas de latência, foi de aproximadamente 7.3 segundos, o que dá bons indícios relativamente à sua eficiência.

```
[23/06/2018 11:43:58] [LOG] CRC verification failed, CRCs dont match!  
[23/06/2018 11:43:58] [LOG] Client [5.249.66.64:56197] sent:  
[23/06/2018 11:43:58] [LOG] 3ffffff  
[23/06/2018 11:43:58] [LOG]  
[23/06/2018 11:43:58] [LOG]  
[23/06/2018 11:43:58] [LOG]
```

Figura 5.8: Validação de integridade através do CRC.

Resultados

```
[23/06/2018 11:24:47] [LOG] Error: Signatures don't match
[23/06/2018 11:24:47] [LOG] Client [5.249.66.64:55993] sent:
ff00
```

Figura 5.9: Adulteração de dados após aplicação de assinatura digital.

Estes resultados permitem concluir que o componente com maior peso no desempenho do protocolo corresponde à fase de transferência de *firmware*, criando assim uma relação de proporcionalidade direta entre a duração da execução do protocolo na implementação atual e o tamanho do ficheiro de *firmware* a ser transferido.

No futuro, o aumento do desempenho do protocolo deverá ter como principal foco a redução do peso deste componente. Para tal podem ser utilizadas várias abordagens: reduzindo o volume de dados a serem transferidos, considerando optar por transferir apenas as diferenças em relação à versão de *firmware* do dispositivo em vez de transferir o ficheiro completo; optando por outras abordagens ao nível da criptografia, que permitissem reduzir o tempo de encriptação e desencriptação.

Em relação à utilização de recursos por parte da máquina onde é executada a plataforma, podemos concluir a partir da visão geral apresentada na figura 5.13 que com o aumento de utilizadores conetados em simultâneo de teste para teste, aumenta também o tempo total necessário para o servidor finalizar a execução do protocolo. Nos casos de teste de 50, 100 e 200 clientes (variações correspondentes às 20:18, 20:22, 20:26), verificou-se um aumento por um fator de 2 em relação ao anterior, i.e., duplicando de teste para teste. A fim de verificar que o protocolo segue o fator de escala mencionado, em termos de tempo de execução, com maior grau de certeza, seria necessária a realização de testes mais exaustivos, abrangendo também máquinas com outras especificações e arquiteturas.

Conclui-se também que os picos máximos de utilização de CPU e RAM não apresentam oscilações significativas quando comparados os casos de teste de 50, 100 e 200 clientes concorrentes, o que dá bons sinais numa perspetiva de escalabilidade horizontal do serviço

Nº clientes	Tráfego de dados do servidor (bytes)		Duração média da sessão (segundos)				
	Recebido	Enviado	Handshake Autenticação + versão	ECDH	Transferência	Total	Max
1 (base line)	5647	685801	0,1930	0,0810	2,8640	3,1380	3,1380
10	10 * base line	10 * base line	0,3850	0,1480	3,6530	4,1860	4,3326
50	50 * base line	50 * base line	0,2640	0,0760	4,4650	4,8050	5,5003
100	100 * base line	100 * base line	0,2000	0,0950	5,2770	5,5720	7,3299
200	200 * base line	200 * base line	0,2000	0,0880	5,4720	5,7600	7,1012

Tabela 5.1: Resultados do teste de escala.

Resultados

```
[23/06/2018 11:58:08] [LOG] VALID SIGNATURE
[23/06/2018 11:58:08] [LOG] IMEI = 01a101a101a101a101a101a101
[23/06/2018 11:58:08] [LOG] Client [5.249.66.64:56299] sent:
.....25425f6b19
.....3a7a2a4c8d
.....
```

(a) Trama assinada com chave privada conhecida

```
[23/06/2018 12:05:06] [LOG] Signatures don't match
[23/06/2018 12:05:06] [LOG] IMEI = 01a101a101a101a101a101a101
[23/06/2018 12:05:06] [LOG] Client [5.249.66.64:56346] sent:
.....9e284a19a9
.....6fa31c90d4
.....;
```

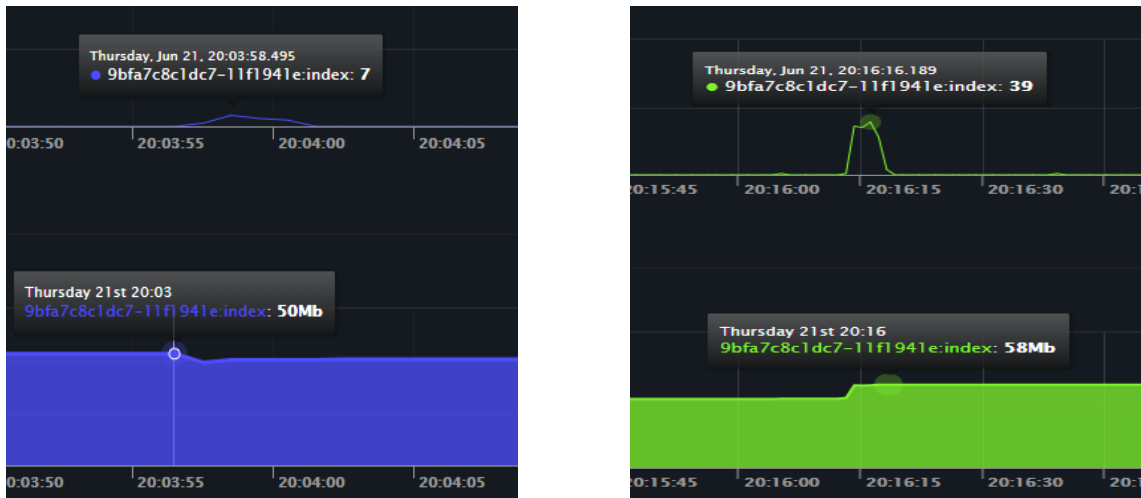
(b) Trama assinada com chave privada desconhecida

Figura 5.10: Assinatura de tramas com diferentes chaves RSA.

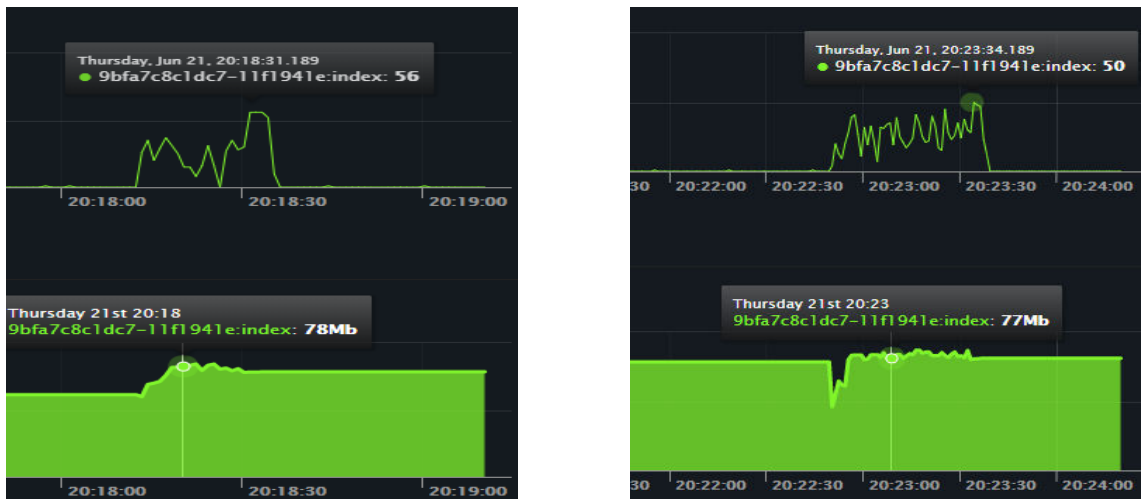
Nº clientes	Desempenho	
	CPU (%)	RAM (MB)
1	7	50
10	39	58
50	56	78
100	50	77
200	54	78

Tabela 5.2: Consumo aproximado de recursos nos vários casos de teste.

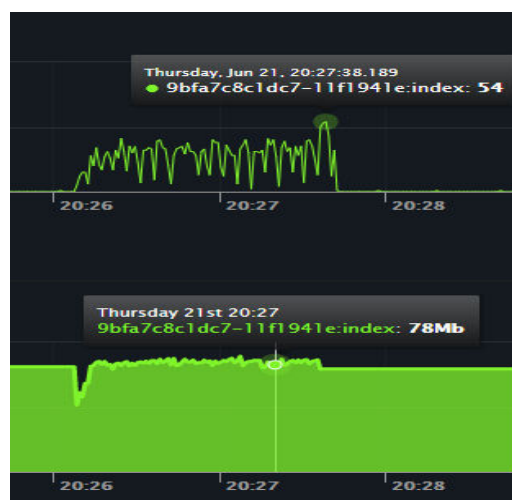
Resultados



(a) Testes com 1 cliente e 10 clientes



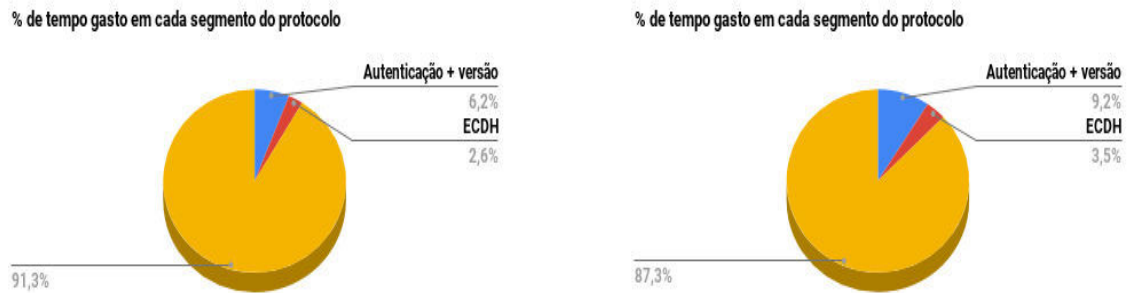
(b) Testes com 50 clientes e 100 clientes



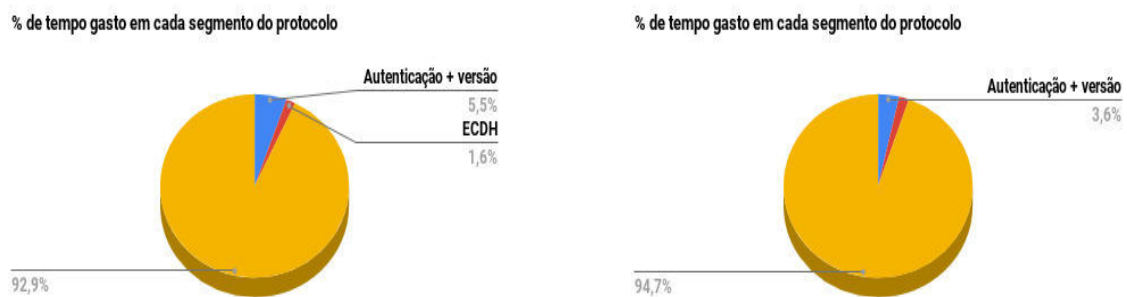
(c) Teste com 200 clientes

Figura 5.11: Utilização de recursos do servidor em vários casos de teste.

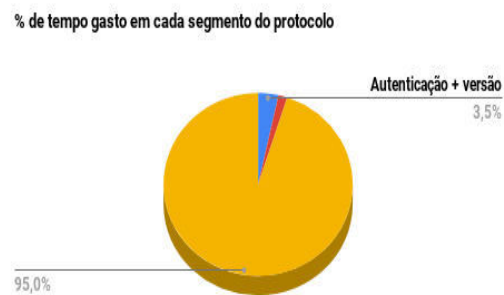
Resultados



(a) Testes com 1 cliente e 10 clientes



(b) Testes com 50 clientes e 100 clientes



(c) Teste com 200 clientes

Figura 5.12: Duração da execução protocolo em diferentes casos de teste.

Resultados

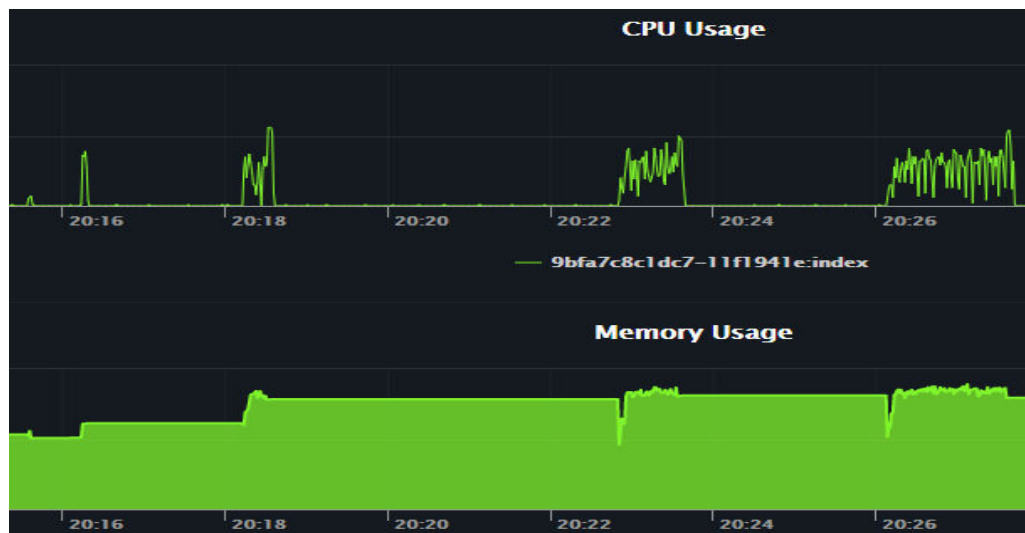


Figura 5.13: Visão geral da utilização de recursos nos cinco casos de teste.

Resultados

Capítulo 6

Conclusões

Neste capítulo é apresentado um resumo do trabalho elaborado, combinado com uma retrospectiva das várias fases de desenvolvimento e avaliação dos objetivos, definidos inicialmente, conjugados com os resultados obtidos. É também apresentado um conjunto de lições aprendidas e potenciais melhorias no âmbito de trabalho futuro, no sentido de apresentar possíveis perspectivas de evolução do protótipo.

6.1 Conclusões

Do trabalho realizado ao longo do período de dissertação resulta um protótipo de uma plataforma de atualizações automáticas de FOTA que soluciona o problema apresentado inicialmente e cuja implementação foi além dos objetivos definidos, representando um *Minimum Viable Product* (MVP) pronto a ser integrado numa fase de testes mais exaustivos para posterior evolução e lançamento em produção. A realização dos testes mencionados permitirá avaliar, com maior grau de certeza, a usabilidade e adaptabilidade da plataforma em contexto real.

Consideram-se garantidas as funcionalidades centrais ao funcionamento da plataforma, as quais permitem realizar e monitorizar uma atualização remota de *firmware* de vários dispositivos em simultâneo. Ao nível do protocolo, acredita-se estarem presentes os principais requisitos relacionados com a segurança e robustez do mesmo, em termos de confidencialidade, integridade e autenticidade dos dados.

Do ponto de vista da implementação, surgiram bastantes desafios em termos da compatibilidade externa com o dispositivo, uma vez que se trata de uma implementação de mais baixo nível, com requisitos mais restritos, resultando na necessidade de reimplementar partes do protocolo ao nível da arquitetura e lógica, nomeadamente na conversão de DH convencional para ECDH. A adaptação para ECDH era um dos melhoramentos considerados face à proposta inicial e apesar de ter sido uma mudança crítica na implementação, acabou por resultar no aumento, já expectável,

do desempenho dessa fase do protocolo. Em conclusão a adaptação do algoritmo demonstrou-se vantajosa para o funcionamento global do protocolo FOTA.

Os resultados obtidos no capítulo anterior permitem concluir que o protocolo é executado de forma eficiente, registando-se valores máximos de execução do protocolo significativamente mais baixos, de aproximadamente 7 segundos, comparativamente com a duração total numa abordagem de atualização manual via intervenção física, que se encontra na ordem dos minutos. Foram também recolhidas métricas que evidenciam indicadores positivos para uma futura escalabilidade horizontal do serviço.

Para além da redução drástica dos tempos de operação associados à atualização de *firmware*, a utilização do protótipo implementado traduz-se também numa poupança de custos para a AddVolt. Numa ótica de escala torna-se inviável ter uma equipa de engenheiros dedicados à operacionalização manual de atualização de *firmware* dos dispositivos, dado que implicaria custos elevados e desviaria recursos de áreas cruciais, e.g., Investigação e Desenvolvimento, Linha de Produto, entre outras.

6.2 Lições aprendidas

Em termos de lições aprendidas durante o desenvolvimento do protótipo, retém-se a necessidade de um bom planeamento da arquitetura do sistema, no sentido de facilitar a integração de novos componentes, concretamente ao nível do tamanho dos campos das tramas, sendo fundamental deixar margem para o incremento dos parâmetros associados a cada campo.

O facto de haver integração com um módulo de mais baixo nível (microcontrolador), com requisitos mais limitados, acarreta a necessidade de uma maior ponderação na escolha de algoritmos, para baixar o impacto em termos de utilização de recursos.

No processo de desenvolvimento de um protótipo deste tipo, é importante configurar ambiente em que vai ser alojado (e.g., servidor na *cloud*) previamente à implementação. Uma configuração tardia pode evidenciar a existência de incompatibilidades com determinadas tecnologias aplicadas. Quanto mais avançada a fase de implementação maior será a probabilidade das incompatibilidades manifestadas implicarem reestruturações mais abrangentes e custosas.

6.3 Evolução futura

Tal como referido no capítulo 4, existiram algumas funcionalidades que não foram consideradas na fase de implementação do protótipo e que deveriam ser desenvolvidas numa fase posterior de melhoria da plataforma. As possíveis melhorias dividem-se em três categorias: *APIs* externas, interface *web* e criptografia.

Na primeira categoria destaca-se a integração com as *APIs* da AWS e da AddVolt a fim de complementar funcionalidades como a autenticação do dispositivo face a uma lista de IMEIs atualizada automaticamente, o carregamento e rastreamento de versões de *firmware* através das *APIs*

Conclusões

referidas. Esta melhoria permitiria a integração com funcionalidades e informação já disponível internamente pela API da Addvolt.

Ao nível da interface *web* seria interessante adicionar uma página de *login* e aperfeiçoar a aparência da interface afim de melhor a experiência do utilizador. A fim de mitigar ataques do tipo MITM, numa fase inicial deveria restringir-se o acesso à interface apenas à rede interna da empresa. Numa fase futura deveria ser também adicionado um certificado que permitisse integrar o protocolo HTTPS.

Em termos de criptografia, seria importante introduzir certificados digitais bem como a aplicação de conceitos como *Key Derivation Function* (KDF) aplicado ao algoritmo ECDH, para obtenção da chave de sessão. Este conceito consiste na utilização de uma função pseudo-aleatória para derivar uma ou mais chaves a partir de um dado valor secreto, podendo ser especificado o seu formato e tamanho. A vantagem da utilização desta técnica reside no fato de ser uma função *one-way*, ou seja, previne que um atacante deduza informações relativas à geração do valor secreto, o que poderia comprometer o protocolo (Paar and Pelzl, 2010).

Conclusões

Referências

AddVolt (2018). Addvolt - produto. Disponível em <http://addvolt.com/pt/>. [Online; acessado 10-Dezembro-2017].

Alliance, O. M. (2012). Open mobile alliance answers industry demand for over-the-air updates. Disponível em <http://openmobilealliance.org/wp-content/uploads/2012/11/August-242006-Open-Mobile-Alliance-Answers-Industry-Demand-for-Over-the-Air.pdf>.

Aschenbruck, N., Bauer, J., Bieling, J., Bothe, A., and Schwamborn, M. (2012). Selective and secure over-the-air programming for wireless sensor networks. In *2012 21st International Conference on Computer Communications and Networks, ICCCN 2012 - Proceedings*.

Augot, D., Batina, L., Bernstein, D. J., Bos, J., Buchmann, J., Castryck, W., Dunkelman, O., Güneysu, T., Gueron, S., Hülsing, A., Lange, T., Mohamed, M. S. E., Rechberger, C., Schwabe, P., Sendrier, N., Vercauteren, F., and Yang, B.-Y. (2015). PQCRYPTO - Post-Quantum Cryptography for Long-Term Security: Initial recommendations of long-term secure post-quantum systems. pages 1–4.

Cisco (2011). Understanding over-the-air provisioning (otap). Disponível em <https://www.cisco.com/c/en/us/support/docs/wireless/4400-series-wireless-lan-controllers/100516-ustnd-otap.html>. [Online; acessado a 15-Novembro-2017].

Daimi, K., Mustafa Saed, S. B., and Rizwan, M. (2016). Securing vehicle ecus update over the air. AICT 2016 : The Twelfth Advanced International Conference on Telecommunications. Disponível em https://www.thinkmind.org/download.php?articleid=aict_2016_3_10_10003. [Online; acessado 15-Janeiro-2018].

ECMA International (2011). *Standard ECMA-262 - ECMAScript Language Specification*. 5.1 edition.

ENISA (2016). *Guidelines for SMEs on the security of personal data processing*. Number December.

Evans, D. (2011). The Internet of Things - How the Next Evolution of the Internet is Changing Everything. *CISCO white paper*, (April):1–11.

Eystein Stenberg, M. (2017). Requirements for ota software updates for connected embedded devices. Disponível em <https://www.embedded.com/electronics-blogs/say-what-/4443238/Requirements-for-OTA-software-updates-for-connected-embedded-devices>. [Online; acessado a 19-Novembro-2017].

REFERÊNCIAS

- Facebook (2018). React - a javascript library for building user interfaces. Disponível em <https://reactjs.org/>. [Online; acessado 20-Junho-2018].
- Foundation, N. (2018). Node.js. Disponível em <https://nodejs.org/en/>. [Online; acessado 22-Junho-2018].
- Francia, G., Ertaul, L., Encinas, L. H., and El-Sheikh, E. (2017). *Computer and Network Security Essentials*. Springer.
- Giry, D. (2017). Bluekrypt keylength - nist recommendations (2016). Disponível em <https://www.keylength.com/en/4/>. [Online; acessado 30-Maio-2018].
- git scm (2018). Git. Disponível em <https://git-scm.com/>. [Online; acessado 22-Junho-2018].
- Gueron, S. (2010). Intel® advanced encryption standard (aes) new instructions set. *Intel Corporation*.
- Inc, D. (2018). Docker - build, ship, and run any app, anywhere. Disponível em <https://www.docker.com/>. [Online; acessado 5-Junho-2018].
- Infineon (2017). Software update over the air (sota). Disponível em <https://www.infineon.com/cms/en/applications/automotive/automotive-security/sotware-update-over-the-air/>. [Online; acessado 17-Janeiro-2018].
- Jr., K. C. M. and Antonakos, J. L. (2009). *Computer Networking for LANS to WANS: Hardware, Software and Security*. CENGAGE learning, First edition.
- Kerry, C. F. and Gallagher, P. D. (2013). Digital signature standard (dss). *FIPS PUB*, pages 186–4.
- Kessler, G. C. (2017). An overview of cryptography. Disponível em [AnOverviewofCryptography](#). [Online; acessado 1-Fevereiro-2018].
- Keymetrics, I. (2017). Keymetrics. Disponível em <https://app.keymetrics.io/>. [Online; acessado 22-Junho-2018].
- libelium (2012). 50 sensor applications for a smarter world. get inspired! Disponível em http://www.libelium.com/50_sensor_applications/. [Online; acessado 15-Dezembro-2017].
- Libelium (2017). Over the air programming - laying the groundwork. Disponível em http://www.libelium.com/downloads/documentation/over_the_air_programming.pdf. [Online; acessado 10-Janeiro-2017].
- Ltd., O. M. A. (2017). Open mobile alliance - about. Disponível em <http://openmobilealliance.org/about-oma>. [Online; acessado 3-Janeiro-2018].
- Matin, M. A. (2006). *Wireless Sensor Networks - Technology and Protocols*. Intech.
- Microsoft (2017). quantum development kit. Disponível em <https://www.microsoft.com/en-us/quantum/development-kit>. [Online; acessado 3-Fevereiro-2018].
- Mironov, I. (2005). Hash functions : Theory, attacks, and applications Theory of hash functions. *Microsoft Research, Silicon Valley Campus*, pages 1–22.

REFERÊNCIAS

- Open Mobile Alliance Ltd. (2007). Firmware update management object architecture (fumo). Disponível em http://www.openmobilealliance.org/release/FUMO/V1_0-20070209-A/OMA-AD-FUMO-V1_0-20070209-A.pdf. [Online; acessado a 10-Janeiro-2018].
- Paar, C. and Pelzl, J. (2010). Introduction to public-key cryptography. In *Understanding Cryptography*, pages 149–171. Springer.
- Randall A. Snyder, M. D. G. (2001). *Wireless Telecommunications Networking with ANSI-41*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition.
- Research, A. (2016). Abi research anticipates accelerated adoption of automotive software over-the-air updates with nearly 180 million new sota-enabled cars shipping between 2016 and 2022. Disponível em <https://www.abiresearch.com/press/abi-research-anticipates-accelerated-adoption-auto/>. [Online; acessado 14-Janeiro-2018].
- Sharon Peleg, R. B. S. (2007). The basics of updating mobile firmware over-the-air (fota). Disponível em https://www.eetimes.com/document.asp?doc_id=1275048. [Online; acessado 20-Novembro-2017].
- Sheffer, Y., Holz, R., and Saint-Andre, P. (2015). Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7525.
- Stallings, W. (2013). *Cryptography and network security: Principles and practice*, sixth edition.
- Statista (2017). Automotive electronics cost as a percentage of total car cost worldwide from 1950 to 2030. Disponível em <https://www.statista.com/statistics/277931/automotive-electronics-cost-as-a-share-of-total-car-cost-worldwide>. [Online; acessado 15-Janeiro-2018].
- Stevens, M., Bursztein, E., Karpman, P., Albertini, A., and Markov, Y. (2017). The first collision for full SHA-1. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10401 LNCS:570–596.
- Stevens, M. M. J. et al. (2012). *Attacks on hash functions and applications*. Mathematical Institute, Faculty of Science, Leiden University.
- Techterms (2016). Ota definition. Disponível em <https://techterms.com/definition/ota>. [Online; acessado a 15-Novembro-2017].
- Weintraub, B. K. and Blunden, Bill, E. (2013). Algorithms, Key Sizes and Parameters Report. *Commentary*, (October):1–5.