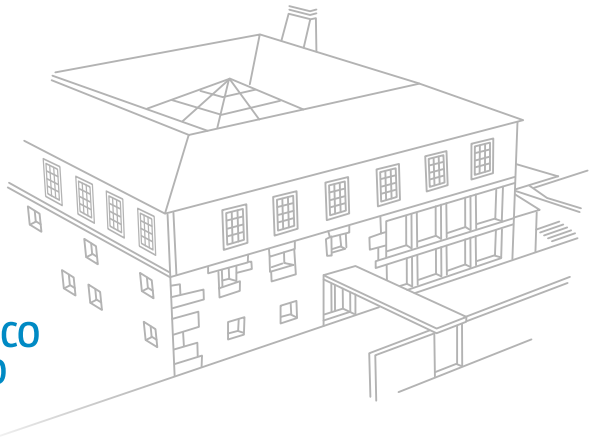

ESTGF | POLITÉCNICO DO PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Aplicação de conversação online com garantia de confidencialidade, anonimato e identidade

DESIGNAÇÃO DO MESTRADO Engenharia Informática

AUTOR Pedro Miguel Oliveira Fernandes

ORIENTADOR(ES) Prof. Doutor António Alberto dos Santos Pinto

ANO 2016

Aplicação de conversação *online* com garantia de confidencialidade, anonimato e identidade

Pedro Miguel Oliveira Fernandes
ESTG-IPP
8080084@estg.ipp.pt

Tese apresentada à Escola Superior de Tecnologia e Gestão
para obtenção do grau de Mestre em Engenharia Informática
realizada sob a orientação do
Prof. Doutor António Alberto dos Santos Pinto
Professor Adjunto na Escola Superior de Tecnologia e Gestão

2016

Agradecimentos

A realização deste projeto de mestrado contou com importantes apoios, sem os quais não se teria tornado uma realidade, pois significa o resultado de um grande esforço.
Ao meu orientador, Professor António Pinto, que, graças à sua experiência, conhecimento, dedicação e profissionalismo, me guiou ao longo desta tese. Realçando a sua disponibilidade na realização de reuniões e acompanhamento, bem como as boas orientações que sempre me concedeu.
Prezo os meus mais sinceros agradecimentos aos meus pais, pelo natural apoio demonstrado e esforço realizado, para que finaliza-se mais uma etapa da minha vida académica. Por último, a todos aqueles que cooperaram positivamente de modo a aprimorar este mesmo projeto. O meu profundo e sentido agradecimento a todos.

Resumo

A Tecnologia de comunicação por proximidade *Near Field Communication* está cada vez mais presente no quotidiano dos utilizadores de *smartphones*. Já é possível trocar-se conteúdos por *Near Field Communication* em vários sistemas baseados em *Android*. Espera-se que esta tecnologia seja uma das com mais relevo em sistemas de pagamentos com *smartphones*. Neste contexto, propõe-se o desenvolvimento de uma aplicação de *chat* segura para *smartphones* com suporte para anonimato e confidencialidade. A aplicação deve suportar a partilha de contactos por *Near Field Communication* e o servidor deve suportar um funcionamento em que o servidor é incapaz de decifrar as mensagens trocadas (*zero knowledge*). É ainda esperado que a aplicação suporte servidores empresariais. A aplicação deverá ser desenvolvida tendo em especial atenção a sua usabilidade, ou seja, com uma *interface* de utilização simples, intuitiva e amiga do utilizador.

Abstract

Communication Technology proximity *Near Field Communication*, it is more and more present in the daily lives of smartphone users. It is possible to exchange content-for *Near Field Communication* in several Android-based systems. It is expected that this technology is one of more payments embossed with smartphones systems. In this context, it proposes the development of a secure chat application for smartphones with support for anonymity and confidentiality. The application must support the sharing of contacts for *Near Field Communication* and the server must support operation in which the server is unable to decipher the messages exchanged (zero knowledge). It is also expected that the application supports enterprise servers. The application should be developed with particular attention to its usability, that is, with a simple user interface, intuitive and user friendly.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Lista de Siglas e Acrónimos

AES Advanced Encryption Standard

API Application Programming Interface

BLE Bluetooth Low Energy

BR/EDR Bit Rate/Enhanced Data Rate

CORS Cross-Origin Resource Sharing

CRUD Create, Read, Update and Delete

CSS Cascading Style Sheets

DOM Document Object Model

ECDH Elliptic Curve Diffie Hellman

ECMA European Computer Manufacturer's Association

EFF Electronic Frontier Foundation

ETSI European Telecommunications Standards Institute

E2E End-to-End

EEEMA End-to-End Encrypting Messaging Application

GCM Google Cloud Messaging

HMAC Keyed-Hash Message Authentication Code

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IGE Infinite Garble Extension

ISO International Organization for Standardization

JSON JavaScript Object Notation

JWT JSON Web Token

KDF Key Derivation Function

MiTM Man-in-The-Middle

MVC Model-View-Controller

NaCl Networking and Cryptography library

NFC Near Field Communication

ORM Object-Relational Mapping

OTR Off-the-Record

PBKDF2 PasswordBased Key Derivation Function 2

PHP Hypertext Preprocessor

REST Representational State Transfer

RF Radio Frequency

RFID Radio Frequency Identification

RIA Rich Internet Applications

RK Root Key

SCIMP Silent Circle Instant Message Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

UDID Unique Device Identifier

URL Uniform Resource Locator

XML Extensible Markup Language

XHTML Extensible HyperText Markup Language

XMPP Extensible Messaging and Presence Protocol

Capítulo 1

Introdução

As tecnologias de comunicação têm vindo a transformar rapidamente e radicalmente a vida e o trabalho das pessoas. As alterações dadas pelas tecnologias de comunicação têm sempre um enorme impacto em todos os aspetos da vida, especialmente sobre a forma como as pessoas interagem umas com as outras. Tecnologias como a Internet e os *smartphones* são uma parte substancial da vida quotidiana. A Internet fez com que novos modelos de interação humana fossem possíveis através do aparecimento de serviços de troca de mensagens, fóruns e redes sociais. Assim, estas interações humanas levaram a haver mais *smartphones* com acesso à Internet do que computadores, embora isso não seja conhecido de forma tão ampla. A penetração móvel global foi de 100% no final de 2015. Em termos gerais, existe um cartão SIM por cada humano no planeta [47].

Os serviços ou aplicações de conversão móvel são hoje ubíquas e desempenham um papel muito importante no quotidiano e na vida social das pessoas, uma vez que, se tem vindo a adaptar aos requisitos humanos, tornando-se mais fáceis de usar, mais baratos e acessíveis. Redes sociais como o *Facebook* ou *Twitter* foram um fenómeno de rápida expansão na última década. O uso destas aplicações é tão avassaladora que transformou a forma de comunicação das pessoas. Nos dias que correm as pessoas gastam cada vez mais tempo nos seus *smartphones* para obtenção de informação, entretenimento, para comunicar ou troca de mensagens com outras pessoas[4].

Decorrente da rápida expansão das redes sociais, surgiu também uma grande adesão a plataformas de conversação *online* de tempo real. Fatores como o alcance global destas novas plataformas de conversão, a sua associação a redes sociais já existentes e o seu baixo custo de utilização, contribuíram para a sua forte aceitação. Naturalmente que um aumento na facilidade de comunicação não acarreta unicamente vantagens já que, quando aliada uma falsa sensação de anonimato, ingenuidade, jovialidade ou a uma má avaliação de risco, podem surgir problemas relacionados com a privacidade, anonimato e com a própria segurança dos intervenientes. Questões estas que podem ser consideradas também sob um ponto de vista de evitar tais problemas mesmo quando o servidor é comprometido.

Recentemente, com o crescente número de ataques e a crescente preocupação com a segurança, serviços de troca de mensagens têm começado a oferecer suporte à cifra *End-to-End* (E2E), com

intuito de proteger o conteúdo transmitido em canais de comunicação inseguros. Cifra E2E tem como finalidade a proteção dos dados transmitidos entre duas extremidades de comunicação e tem como objetivo o evitar de fugas de informações por meio da intervenção de terceiros. Deste modo, existe um nível de proteção extra, em que os dados só podem ser lidos com as chaves de cifra e decifra [19].

Não existe nenhuma maneira perfeita para garantir qualquer serviço ou aplicação. É sempre um *trade-off*, processo imparável, não um ato *one-off*. O que hoje pode ser seguro, amanhã pode não o ser. É importante que se reavalie continuamente todas as práticas de segurança implementadas [29]. Segundo o relatório *NowSecure Mobile Security Report*: um quarto das aplicações móveis de troca de mensagens incluem uma ou mais falhas de segurança de alto risco, 35% das comunicações enviadas por *smartphones* não são cifradas, 43% dos utilizadores não tem um PIN, padrão de bloqueio ou código de acesso, e 50% dos *smartphones* ligam-se a uma rede sem fios desprotegida, pelo menos uma vez por mês [29].

A preocupação com a privacidade, é um tema bastante controverso, pois aborda um assunto de grande interesse e inquietação da sociedade. A privacidade é a habilidade de uma pessoa em controlar a exposição ou disponibilidade de informações sobre si, e se relaciona com a capacidade de existir na sociedade de forma anónima podendo se usar de uma identidade falsa. Os constantes escândalos de invasão de privacidade e espionagem, que chegam a envolver chefes de estado, trouxeram maior evidência para o assunto, mostrando ao mundo que a privacidade é muito reduzida.

O crescente uso das aplicações de troca de mensagens, tem gerado desafios para os direitos fundamentais dos utilizadores, a saber: o direito à privacidade, à liberdade de expressão e a liberdade de associação. O grande volume de informações pessoais que são publicadas diariamente colocam em risco tais direitos.

Em 1999 Scott Mcneally, o CEO da *Sun Microsystems*, estendeu declarações polémicas que afirmam inexistir privacidade quanto aos dados pessoais armazenados nos serviços [10]. No entanto, cada vez mais as pessoas esperam das empresas e das aplicações desenvolvidas uma maior segurança da informação. Na Europa o direito à proteção dos dados pessoais é um direito fundamental da Carta Europeia dos Direitos Fundamentais. Esta lei aplica-se a todos os membros da União Europeia. Diante disto, e cada vez mais existe a necessidade dos servidores que suportam serviços ou aplicações de troca de mensagens E2E não sejam capazes de aceder aos dados ou mensagens trocadas entre utilizadores.

No que tange ao último dos problemas referenciados, o anonimato, é presentemente uma tendência que se tornou realmente um grande ponto de interesse pelos utilizadores das aplicações de conversação. Tanto confiança como anonimato são ambas propriedades desejáveis em qualquer tipo de aplicação, até mesmo na Internet. Quando começaram a emergir as primeiras aplicações para *smartphones* em que os utilizadores poderiam usar anonimamente, todos estes ficaram impressionados. No entanto, este é o lugar onde utilizadores mal intencionados tendem a tirar proveito. Isto porque é fácil o envio de mensagens ofensivas e envio de rumores desagradáveis sobre alguém anonimamente.

O anonimato possibilita a utilização de falsas identidades. A utilização de uma identidade diferente da real é vulgar nas aplicações de troca de mensagens, em que o objetivo pode mesmo ser o de assumir uma nova identidade, muitas vezes do sexo oposto. Mas isso também permite que, por exemplo, as crianças possam passar por adultas e aceder a serviços não adequados à sua idade, ou ficarem expostas a esquemas obscuros. Consequentemente estas falsas identidades assumem outras proporções quando se tentam passar por outros indivíduos.

1.1 Objetivos do Trabalho

A facilidade das comunicações digitais entre as pessoas atualmente é desmedida, e o problema de utilização de falsas identidades, relacionado com o anonimato, conduziu a que a principal motivação deste projeto assenta-se numa resolução para esta adversidade. Assim, o objetivo deste projeto edifica no desenvolvimento de uma aplicação que crie um meio de comunicação seguro que possibilite a troca de mensagens entre utilizadores de forma segura, privada, anónima perante o servidor. A aplicação deverá ainda garantir, aos intervenientes em cada conversão, a identidade dos mesmos.

1.2 Resultados relevantes

A solução proposta deverá, em particular, garantir:

1. **Identidade:** A solução proposta deverá garantir, aos intervenientes em cada conversão, a identidade dos mesmos.
2. **Confidencialidade:** A solução proposta não deverá ser capaz, em circunstância alguma, de aceder ao conteúdo das mensagens trocadas entre os utilizadores.
3. **Anonimato:** A solução proposta deverá garantir o anonimato dos utilizadores perante o servidor.
4. **Usabilidade:** A solução proposta deverá ter um *interface* de utilização simples, intuitiva e amiga do utilizador.
5. **Portabilidade:** A solução proposta deverá ser integrável para o máximo de plataformas móveis, como fundamentais *Andorid* e *IOS*.

1.3 Estrutura do relatório

O relatório para além do presente capítulo a introdução, encontra-se dividido em outros cinco capítulos, que têm como objetivo apresentar o âmbito do tema deste projeto. Desta forma, no segundo capítulo será apresentada as tecnologias de comunicação por proximidade, linguagens

e *frameworks* de maior pertinência para o projeto, apresentando características e propriedades fundamentais de cada um. No terceiro capítulo, serão analisadas algumas das aplicações de troca de mensagens, relacionadas com o projeto a desenvolver. Estas aplicações têm bastante relevância no mercado, bem como uma vasta quantidade de utilizadores, pois todas permitem suporte a cifra. A análise das várias aplicações é realizada segundo a *Electronic Frontier Foundation* (EFF) *Secure Messaging Scorecard*, que visa dois principais pontos: segurança e usabilidade. Para além disto, será aprofundado cada protocolo implementado por cada aplicação analisada.

O quarto capítulo, tem como finalidade fazer uma apresentação da solução implementada neste projeto, onde integra a caracterização geral da solução. Sendo que, são expostas as funcionalidades e ferramentas aplicadas para a realização dessas mesmas funcionalidades. Nesta secção, é apresentado o modelo do projeto e a sua arquitetura para melhor compreensão da mesma. No que concerne ao quinto capítulo, contém a avaliação da solução e cada um dos objetivos que foram implementados, bem como a sua apreciação. No sexto e último capítulo, contém as conclusões deste trabalho, assim como o trabalho futuro a realizar com intuito de melhorar e refinar este protótipo.

Capítulo 2

Comunicação, Linguagens e *Frameworks*

As tecnologias de comunicação disponíveis hoje em dia na generalidade dos *smartphones* são várias. Tipicamente, cada uma tem um propósito particular que faz com que os fabricantes de equipamentos optem por as incluir. Duas tecnologias de comunicação em proximidade que têm ganho relevo recentemente são o *Near Field Communication* (NFC) e o *Bluetooth*. É frequente encontrarem-se aplicações móveis que façam uso destas tecnologias. *Android Pay*, é um sistema de pagamento *tap-to-pay* que combina a facilidade de utilização e segurança, poupando tempo ao utilizador de andar em torno de cartões ou PINs. *InstaWiFi*, permite aos utilizadores a partilha de rede *Wi-Fi*, através de uma *tag* NFC ou QR *code*. *Bluetooth File Transfer*, dos mais utilizados para transferência de ficheiros via *Bluetooth*. *App Sender Bluetooth*, atua para partilha de aplicações instaladas com outros utilizadores, sem uso de Internet.

O desenvolvimento aplicacional para a *web* e *smartphones* está cada vez mais facilitado com a eclosão de plataformas de desenvolvimento, ou *frameworks*, que possibilitam o desenvolvimento rápido e multi-plataforma. Algumas para desenvolvimento mais genérico, como é exemplo o *Ionic* ou *AngularJS*, outras mais específico como são exemplo o *JSON Web Token* e o *Satellizer*. Estas *frameworks* são construídos sobre linguagens já existentes de que são exemplo o HTML5 e o *Javascript*.

É usual também que aplicações para *web*/*smartphones* requeiram uma componente servidor. Mesmo para este caso, i.e. desenvolvimento *server-side*, encontram-se cada vez mais *frameworks* que agilizam o seu desenvolvimento. O *Laravel* é um exemplo de tais *frameworks*.

2.1 Tecnologias de Comunicação por Proximidade

O NFC e o *Bluetooth* são tecnologias que possibilitam a comunicação de proximidade que se julgam ser as que se irão tornar as adotadas pela generalidade das aplicações, fomentando a sua disponibilidade generalizada.

2.1.1 *Near Field Communication*

NFC é uma tecnologia de proximidade que surgiu como uma extensão do *Radio Frequency Identification* (RFID) e que se encontra em grande expansão. Esta tecnologia possibilita a rápida comunicação entre *smartphones*, usando ligação sem fios de curto alcance. Em NFC a comunicação para partilha de informações requer uma distância inferior a 4 centímetros e com débito máximo de 424 Kbps. O NFC opera a uma frequência de 13,56 MHz.

O uso de NFC torna possível efetuar a ligação de dispositivos eletrónicos e iniciar serviços com um simples toque [16]. O NFC visa auxiliar, ou simplificar, o processo de interligação de *smartphones*, possibilitando um acesso mais intuitivo a serviços como a trocas de conteúdos digitais, pagamentos, descoberta de equipamentos, sincronização de informações e até simplificar o uso diário de transportes e outros serviços públicos [1]. A tecnologia NFC permite interações bidirecionais simples e seguras entre dispositivos eletrónicos, complementando outras tecnologias sem fios [42].

A tecnologia NFC tenta servir de elo de ligação entre as diversas tecnologias sem contacto que existem atualmente. Apresenta como vantagens [8, 16] o facto de ser uma tecnologia aberta, intuitiva, versátil e interoperável. O NFC e as camadas subjacentes da tecnologia NFC seguem normas e implementações universais (*International Organization for Standardization* (ISO), *European Computer Manufacturer's Association* (ECMA), *European Telecommunications Standards Institute* (ETSI)). É intuitivo já que as interações NFC não requerem mais do que a simulação de um simples toque. É versátil porque pode ser utilizado numa ampla gama de ambientes. É interoperável com tecnologias de cartões *contactless*[1]existentes.

Modos de operação

Os *smartphones* que incluem a tecnologia NFC integrada, possibilitam dois modos de operação: o modo ativo e o modo passivo [55]. Um *smartphone* em modo ativo gera a sua própria portadora, resultando daí o seu campo *Radio Frequency* (RF) para fins de transmissão. Necessita de uma fonte de energia para a operação. Deste modo dois *smartphones* com NFC ativos podem alternativamente, gerar campos de RF no sentido de formarem um *link* de comunicação, para transferência de dados. Um *smartphone* com NFC a operar em modo passivo, contém informações que outros *smartphones* podem ler, mas não recolhe nenhuma informação em si. Por exemplo, um *smartphone* em modo passivo funciona como um sinal numa parede. Outros podem ler as informações, mas o sinal não faz mais nada para além da transmissão de informações para outros *smartphones* autorizados.

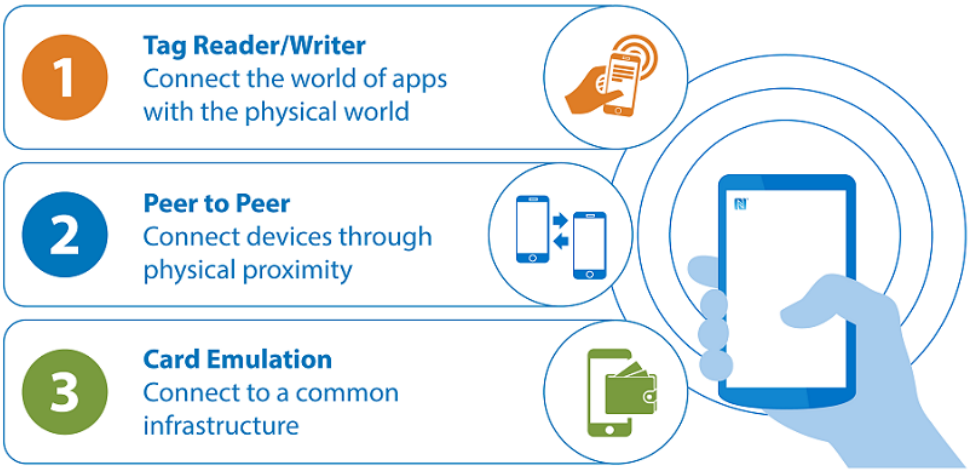

FIGURA 2.1: Modos de comunicação da tecnologia NFC [17]

Modos de comunicação

Na Figura 2.1 estão representados os três modos de comunicação em NFC: o modo leitura/escrita, o modo *peer-to-peer* e o modo emulação de cartão. O modo de leitura/escrita permite que os *smartphones* com tecnologia NFC possam ler informações armazenadas em *tags* NFC. Estas *tags* são incluídas em cartazes, entre outros, e são utilizadas como ferramenta de marketing nas empresas. O *smartphone* funciona no modo ativo para ler o conteúdo de uma *tag*. Quando deteta duas ou mais *tags*, ele baseia-se no algoritmo *anticollision* para selecionar apenas uma *tag*. O *smartphone* também pode escrever dados para uma *tag*. Para gravar os dados numa *tag*, deve conter uma aplicação de escrita, como por exemplo *TagWriter*.

O modo *peer-to-peer* possibilita a troca de informações entre dois *smartphones*. Por exemplo, os utilizadores podem partilhar parâmetros de configuração de redes *bluetooth* ou *Wi-Fi*, trocar cartões de visita virtuais ou fotos em formato digital, entre outros.

O modo emulação de cartão permite que *smartphones* operem como cartões, permitindo aos seus utilizadores realizar transações. No modo de emulação do cartão, o *smartphone* comunica com um leitor externo muito parecido com um cartão inteligente sem contacto. Isso permite pagamentos sem contacto e a emissão de bilhetes por *smartphones*, sem alterar a infra-estrutura existente. O uso do *smartphone* como cartão de crédito, de débito, de acesso, entre outros, é possível.

2.1.2 *Bluetooth*

Bluetooth é uma tecnologia de proximidade para troca de dados a curtas distâncias, que visa a substituição dos cabos de conexão. Contém várias caraterísticas como robustez, baixo consumo

[1]Cartões de pagamento com tecnologia de leitura por aproximação (*contactless*) permitem a realização de operações de pagamento através da aproximação do cartão, a curta distância, de um terminal de pagamento automático.

FIGURA 2.2: Versões *Bluetooth* [26]

de energia e baixo custo. A tecnologia *Bluetooth* foi concebida em primeiro lugar para suportar dispositivos com a tecnologia *wireless*, incluindo telefones sem fios, teclados, entre outros [39].

A comunicação entre dispositivos *Bluetooth* é de curto alcance, usando redes *ad hoc* conhecidas como *piconets*. *Piconet* é uma rede de dispositivos ligados por *Bluetooth*. A dimensão de uma *piconet* varia entre dois e oito dispositivos. Quando é estabelecida, um dispositivo assume o papel de *master*, enquanto todos os outros dispositivos agem como *slaves*. *Piconets* são estabelecidos de forma dinâmica e automática, conforme os dispositivos *Bluetooth* entram e saem do raio de alcance do *master*.

O *Bluetooth* define essencialmente duas formas de operação: o *Bit Rate/Enhanced Data Rate* (BR/EDR) e o *Bluetooth Low Energy* (BLE). O BR/EDR surgiu na versão do *Bluetooth* mais antiga, que começou com a versão 1.0, e que tem evoluído ao longo do tempo. Já o BLE é a variante do *Bluetooth* que opera com baixa potência e que foi introduzido com a versão 4.0. A Figura 2.2 mostra as combinações possíveis de configuração de entre as versões *Bluetooth* disponíveis.

BR/EDR

O BR/EDR permite que o *Bluetooth* efetue *streaming* de multimédia de alta qualidade, otimizado para envio de um fluxo constante de dados de alta qualidade e eficiente em termos de consumo de energia. O modo *Bit Rate* suporta uma taxa de 1 Mbps enquanto que o modo *Enhanced Data Rate* suporta uma taxa de 2 Mbps. O modo BR/EDR foi projetado para que dispositivos como ratos e teclados sem fios tenham uma autonomia superior (até 5 vezes). Para cenários de conexão que requerem a interação do utilizador, a versão 2.1 oferece proteção *Man-in-The-Middle* (MiTM)[2], que elimina a possibilidade de um terceiro intercetar informações. As melhorias de comunicação na versão 2.1 permitem ainda o uso de NFC [38].

BLE

O BLE foi desenvolvido para ser utilizado por dispositivos que funcionam durante longos períodos de tempo. O BLE introduz um novo modo de espera, ou seja, BLE permanece em modo suspensão constantemente exceto quando uma conexão é iniciada. Este modo é chamado de *ultra-low peak*, que reduz o consumo de energia e que lhe dá a capacidade de funcionar durante

[2]É quando um atacante faz conexões independentes com as vítimas e retransmite mensagens entre eles para os fazer acreditar que estão a comunicar uns com os outros, quando, na verdade toda a conversação é controlada pelo atacante.

largos períodos de tempo em baterias tipo *coin-cell*. Assim como *Bluetooth*, BLE opera na banda 2.4 GHz ISM. Introduz também o suporte para a cifra de dados com *Advanced Encryption Standard* (AES) de 128 bits [40].

Em resumo, *Bluetooth* e BLE, são utilizados para fins muito diferentes. *Bluetooth* pode lidar com uma grande quantidade de dados, mas consome bastante bateria de forma rápida e com um custo superior. BLE é utilizado para aplicações que não necessitam de trocar grandes quantidades de dados, e podem, portanto, funcionar com a energia da bateria durante anos a um custo mais baixo. Tudo depende do que se está a tentar realizar.

2.2 Linguagens

Linguagens como o HTML5 e o *Javascript* são frequentemente utilizadas no desenvolvimento de páginas *web*, serviços *web* e até aplicações móveis.

2.2.1 HTML5

HTML5 é a quinta revisão de *Hypertext Markup Language* (HTML), uma linguagem padrão utilizada para descrever o conteúdo e a aparência de páginas *web*. HTML5 foi aprimorado para resolução de problemas de compatibilidade que afetam a versão anterior do HTML. Uma das maiores diferenças entre o HTML5 e as versões anteriores, é que as versões mais antigas do HTML requerem *plugins* proprietários e *Application Programming Interface* (API)s [33]. Por isso, uma página *web* que foi construída e testada num determinado *browser* pode não carregar corretamente num outro *browser*.

Na versão 5 foi especificado um *Document Object Model* (DOM) mais complexo e completo do que o de HTML4 ou da *Extensible HyperText Markup Language* (XHTML) [33]. Este DOM define todos os nomes e atributos dos componentes que compõem uma página. Torna também possível a sua manipulação por intermédio de código *JavaScript*. Fornece uma interface comum para tornar os elementos de carregamento de forma inteligível. Por exemplo, não há necessidade de instalar um *plugin Flash* em HTML5 porque o elemento será executado por si só.

Uma das novidades do HTML5 é o suporte melhorado para a representação de conteúdos multimédia em dispositivos móveis. Introduzindo novos recursos sintáticos para suportar este processo, tais como *tags* de vídeo e áudio. HTML5 introduz inclusive novos recursos que mudam a forma como os utilizadores interagem com os documentos [20]. O HTML5 inclui ainda outros benefícios como [15]: *built-in video/audio playback*, cache *offline*, código mais limpo, *browser cross-compatibility*, otimização móvel, geolocalização, base de dados *offline* e *great forms*. Em versões anteriores do HTML, *webmasters* eram forçados a recorrer a programas de terceiros, como o *Adobe Flash Player*, o *Quicktime* ou o *Silverlight*, para reproduzir vídeo. Este era um método confuso mas permitia a reprodução multimédia baseado na *web*. Resultava muitas vezes em erro ou dificuldades acrescidas. O HTML5 resolveu este problema com suporte nativo para

vídeo e áudio. O HTML5 permite ainda a utilização de *cache offline*, i.e. os utilizadores podem carregar determinados elementos numa página *web* sem conexão ativa com Internet, desde que tais elementos estejam armazenados em *cache* local.

HTML5 foi pensado para aumentar a sua legibilidade, não apenas para programadores, mas também para melhorar o seu processamento por motores de pesquisa. É compatível com os principais *browsers*, incluindo *Firefox*, *IE*, *Chrome*, *Safari* e *Opera*. Tal não significa que todos estes *browsers* suportem todos os novos elementos do HTML5, mas no mínimo devem ser capazes de ler o *Doctype*. Foi ainda pensado para a facilitar a criação de sites móveis e aplicações. *Responsive websites* são facilmente construídos utilizando HTML5, oferecendo uma funcionalidade semelhante em todos os tipos de dispositivos. O suporte para a geolocalização é um dos benefícios do HTML5. A localização passa a estar diretamente acessível em qualquer *browser* ou aplicação compatível. O HTML5 contém também uma base de dados em *SQL*, que permite a quem está no lado cliente armazenar dados nos seus próprios computadores (base de dados *offline*) [15].

A aceitação de HTML5 depende deste ser suportado pelos *browsers*. Como só é suportado nas versões mais recentes dos *browsers*, tal poderá ser visto como uma limitação. O HTML tem componentes bastante estáveis ao longo do tempo, contudo a própria linguagem é considerada como um trabalho em curso. Tal significa que qualquer um dos seus elementos pode mudar.

2.2.2 *JavaScript*

JavaScript é uma linguagem de programação interpretada, de alto nível, dinâmica e *untyped* [3]. O *JavaScript*, em conjunto com HTML e *Cascading Style Sheets* (CSS), é uma das principais tecnologias de produção de conteúdos para a *World Wide Web*. É uma linguagem de *script* multi-paradigma com base num protótipo dinâmico. Têm suporte para programação funcional, orientada a objetos e imperativa. É a linguagem de programação que executa do lado do cliente (*client-side*) mais utilizada. É o *browser* que suporta a carga de processamento [9].

O *JavaScript* apresenta como principais vantagens [25]: a independência entre cliente e servidor, velocidade, uma programação baseada em eventos, a independência de plataforma e *interfaces* mais ricas.

Ao existir uma separação das camadas cliente e servidor, de forma que eles se comuniquem apenas via API (*Representational State Transfer* (REST), por exemplo), é criada uma independência total. É então mais fácil separar as áreas de *back-end* e *front-end*. Em *JavaScript* estar do lado cliente, torna-se mais rápido porque quaisquer funções de código podem ser executadas imediatamente, em vez de ter de comunicar com o servidor e esperar por uma resposta.

JavaScript é igualmente uma linguagem baseada em eventos, isso significa que segmentos de código diferentes são executados quando ocorre determinado evento. *JavaScript* é uma plataforma de linguagem independente. Qualquer *browser* habilitado com *JavaScript* pode interpretar o mesmo. Posto isto, as suas *interfaces* são mais ricas, pois incluem itens como componentes *drag-and-drop* e *sliders* que dão uma *interface* mais rica para os utilizadores [25].

JSON

JavaScript Object Notation (JSON) é a notação de dados adotada pelo *JavaScript*. É baseada num formato de texto que é completamente independente do idioma, mas faz uso de convenções que são familiares aos programadores da família-*C*, incluindo *C*, *C++*, *C#*, *Java*, *JavaScript*, *Perl*, *Python*, e muitas outras [23].

Essencialmente, JSON é um método de leitura fácil utilizado para armazenar matrizes e objetos com valores como *strings*. É usado para a transferência de dados e é muito menos detalhado do que algumas das outras linguagens, como *Extensible Markup Language* (XML). Usualmente, JSON é usado quando o *front-end* da aplicação requer alguns dados do *back-end*, sem ter que recarregar a página.

JSON pode ser descrito como uma forma de armazenamento de informações organizada e de fácil acesso. Oferece uma coleção de dados legível, acessível de forma lógica. JSON permite superar o problema do *cross-domain*, pois possibilita o uso do método chamado JSONP (*Padding*), i.e. utiliza uma função de *callback* para enviar os dados JSON de volta ao nosso domínio. Esta é uma das capacidades que faz com que JSON seja útil, pois revela um monte de portas que antes eram difíceis de contornar [23].

Como características chave do formato JSON [7, 31], pode-se considerar o facto de ser simples, rápido, com suporte para vários tipos de dados com disposição, *mapping* e *parsing*. É muito mais simples, tem uma gramática e mapas muito menores, mais diretamente ligados às estruturas de dados. JSON não é uma linguagem de marcação de documentos, por isso não é necessário definir *tags* ou atributos para representar dados. JSON é mais rápido quando comparado com XML, pois os dados estão *serialized*. Suporta tipos de dados como inteiros, *strings* e *arrays*. É ainda orientado a dados (*mapping*). JSON utiliza apenas *eval()* para *parsing*, ou seja, para interpretar o código *JavaScript* e retornar o resultado.

2.3 *Frameworks*

Frameworks como *Ionic*, *AngularJS*, *JSON Web Token* ou *Satellizer* possibilitam o desenvolvimento rápido e multi-plataforma para o contexto *web/smartphone*. Estas *frameworks* são construídas sobre linguagens já existentes de que são exemplo o HTML5 e o *Javascript*. Uma eventual componente servidor para tais aplicações pode ser também construída com base em *frameworks* como o *Laravel*.

2.3.1 *Laravel*

Laravel é uma *framework open-source* de desenvolvimento *web* escrita em *Hypertext Preprocessor* (PHP). Foi criada por Taylor Otwell e segue a arquitetura padrão *Model-View-Controller* (MVC) (ver Figura 2.3) [44]. *Laravel* possibilita o desenvolvimento modular de código. Foi

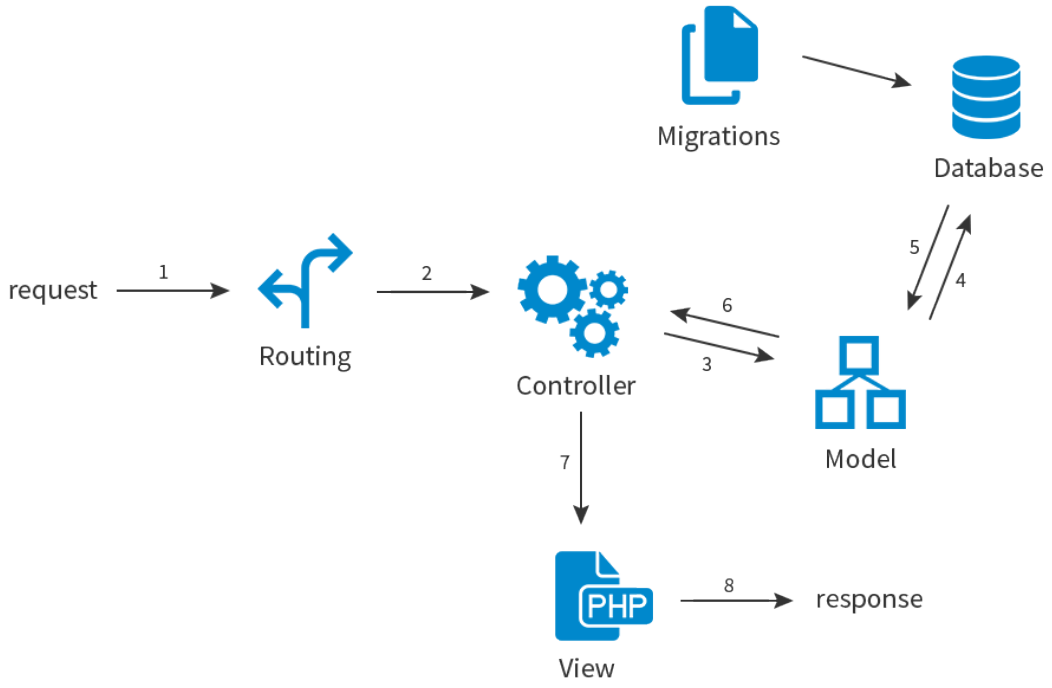

FIGURA 2.3: Representação do fluxo de MVC em *Laravel* [41]

projetada para melhorar a qualidade do *software* e para reduzir custos de desenvolvimento de manutenção. *Laravel* é exemplo de uma nova geração de *web frameworks*.

A maioria dos *websites* requer um conjunto comum de funcionalidades como a gestão de utilizadores, manipulação de sessões ou a validação dos dados introduzidos, entre outras. Uma *framework* é algo que já inclui tais funcionalidades, não requerendo que estas sejam implementadas de raiz sempre que se desenvolve um novo *website*.

Laravel, foi concebido de forma a projetar uma solução que contenha o mesmo número de objetivos mas com menos código. Não importa o quão único seja um sistema, *Laravel* é flexível o suficiente para trabalhar. *Laravel* foi projetada para atingir o *sweet-spot* entre minimalismo e funcionalidade. A implementação de soluções em *Laravel*, é de forma limpa, simples e elegante [32].

O *Laravel* inclui uma interface de linha de comandos denominada de *artisan*. O *artisan* permite que se executem tarefas de administração e manutenção, bem como de testes de funcionalidades e de excerto de código. A execução de migrações sobre a base de dados, a execução de testes unitários, ou a execução de tarefas programadas, são exemplo de tarefas de manutenção que podem ser executadas recorrendo ao *artisan* [28].

Uma aplicação desenvolvida em *Laravel* deve adoptar uma filosofia MVC. Esta filosofia é conseguida em *Laravel* devido ao *workflow* de funcionamento das aplicações desenvolvidas em *Laravel*. A Figura 2.3 representa este *workflow* que compreende os seguintes elementos [41]: *routing*, *controller*, *view*, *model*, *database* e *migrations*. O *routing* representa o conjunto de pontos de acesso

disponibilizados pela aplicação. Em concreto, contém o conjunto de *URLs* para as componentes da aplicação que estão visíveis, bem como o tipo de pedido *Hypertext Transfer Protocol* (HTTP) (*get, post, ...*) suportado por estes componentes. Este recurso é orientado como micro-*frameworks*, tais como *Sinatra* (*Ruby*) e *Silex* (PHP).

O *controller* representa todos os comportamentos da aplicação. Estes comportamentos são baseados em classes do tipo *controller*. Métodos que se encontrem nas classes *controller* podem carregar uma *view* estática ou fazer pedidos de dados à base de dados. Se uma *view* é estática, são efetuados os passos 1-2-7-8 da Figura 2.3. Se não for estática, e existir um pedido de dados, então continua para o passo 3.

O *model* é utilizado para a comunicação com a base de dados através de *ActiveRecord*. À implementação em *Laravel* do *ActiveRecord* chama-se *Eloquent*. Esta facilita a comunicação com a base de dados. Cada tabela na base de dados tem o seu próprio modelo. *Database* representa todo o conjunto de comandos para a configuração da base de dados, que se encontra localizada em *config/database.php*.

Migrations retratam ficheiros para a construção de *schemas* de base de dados. Através de uma *migration* é possível fazer alterações rápidas no *schema* da base de dados, sem acesso direto a esta. A *view* é conseguida com ficheiros *template* aos quais são adicionadas palavras chave com uma sintaxe específica (*blade*). Possibilita a construção modular de vistas da aplicação.

O *Laravel* apresenta como características principais [37]: a modularidade, a testabilidade, a definição de pontos de acesso, a gestão de configurações, a representação de bases de dados relacionais sob a forma de objetos ou *Object-Relational Mapping* (ORM), o suporte para migrações da bases de dados, a existência de um motor de *templates* bem como o suporte para o envio de *emails* e autenticação, entre outras.

A *framework Laravel* utiliza 200 bibliotecas diferentes e é dividida em módulos individuais. Com a integração do *Composer Dependency Manager*, o suporte para a modularidade é conseguido e possibilita a atualização de um ou mais módulos. A testabilidade é nativa ao *Laravel* pois já inclui as ferramentas necessárias para proceder a teste unitários ao código desenvolvido. Na maioria das vezes, é esperado que uma aplicação possa ser executada em diferentes ambientes (desenvolvimento, teste, produção). Para tal, o *Laravel* permite definir configurações para ambiente diferentes e, em seguida, seleciona de forma automática as configurações corretas, dependendo do contexto de execução da aplicação.

A representação de bases de dados relacionais sob a forma de objetos ou ORM, permite emitir consultas na base de dados com sintaxe PHP, onde é possível encadear métodos em vez de escrever *SQL*. *Eloquent*, ajuda a definir modelos interligados, semelhante ao que se encontra em *Ruby on Rails*. Sendo que, ORM é compatível com diferentes bases de dados, como *PostgreSQL*, *SQLite*, *MySQL* e *SQL Server*. No que respeita ao suporte para migrações da bases de dados, inspirado por *Rails*, estas caraterísticas permitem definir com código PHP o esquema da base de dados, e manter todas as mudanças com as migrações da base de dados. A existência de

um motor de *templates* torna-se num modelo de linguagem leve, com a qual se pode criar uma hierarquia de *layouts* com blocos pré-definidos, onde o conteúdo dinâmico é injetado.

O suporte para envio de *emails* é conseguido com a biblioteca *SwiftMailer* e a sua classe *mail*. *Laravel* fornece, de base, todo o material necessário para o registo e autenticação de utilizadores. Tal facilita a implementação da autenticação, uma funcionalidade muito comum na generalidade das aplicações *web*. *Laravel* integra vários serviços de *queue*, como a *Amazon*, *SQS* e *IronMQ*, para poder atrasar tarefas que consomem bastantes recursos, como por exemplo o envio de *emails* para um grande número de utilizadores [37].

2.3.2 JSON Web Token

JSON Web Token (JWT) é um *open standard* (*RFC* 7519), que define uma forma compacta e auto-suficiente para transmissão de informações seguras entre as partes e sob a forma de um objeto JSON. Um JWT pode ser assinado utilizando uma chave secreta (com algoritmo *Keyed-Hash Message Authentication Code* (HMAC)) ou recorrendo a um par de chaves pública/privada usando RSA. Tal torna a informação em informação confiável que pode ser verificada.

O JWT foi desenvolvido para ser compacto e independente. Sendo compacto, pode ser enviado no próprio *URL*, como um parâmetro de um *post* ou dentro do cabeçalho HTTP. Sendo compacto também leva a um tamanho reduzido e a uma transmissão rápida. Sendo independente, evitam-se consultas adicionais a bases de dados. Esta independência é conseguida incluindo todas as informações necessárias sobre o utilizador no objeto JSON [5].

O *Single Sign On* é uma estratégia de autenticação que utiliza amplamente JWT, devido a esta ter pouco impacto na sobrecarga geral do sistema e ao fato de ser de fácil utilização entre sistemas em domínios diferentes. JWT permite transmitir informações com segurança entre as partes. Com a assinatura digital da informação pode-se assegurar que o remetente é quem diz ser. Além disso, como a assinatura é calculada utilizando o *header* e o *payload*, pode-se verificar se o conteúdo foi ou não alterado [5].

Na Figura 2.4 pode observar-se o funcionamento de JWT. Inicialmente, quando é efetuada a autenticação pelo utilizador com as suas credenciais de acesso, um JWT é devolvido para ser guardado localmente (em vez da tradicional abordagem de criação de uma sessão no servidor e retorno de *cookie*). Sempre que um utilizador aceder a uma zona protegida, deve enviar o JWT no cabeçalho HTTP, utilizando o *Bearer schema* [5].

Este é um mecanismo de autenticação sem estado, ou *state-less*. No acesso a zonas protegidas, o servidor verifica se existe um JWT válido no cabeçalho HTTP e, se existir, é dada permissão ao utilizador. Como JWTs são *self-contained*, toda a informação necessária está lá. Tal reduz à necessidade de acessos à bases de dados. Não importa quais os domínios que estão a servir as APIs, com *Cross-Origin Resource Sharing* (CORS)[3] não existe qualquer problema, pois não faz uso de *cookies* [5].

[3]É um mecanismo que permite que um ou mais recursos de acesso restrito (como exemplo, fontes) numa página *Web*, possa ser solicitado de outro domínio fora do qual este foi originado.

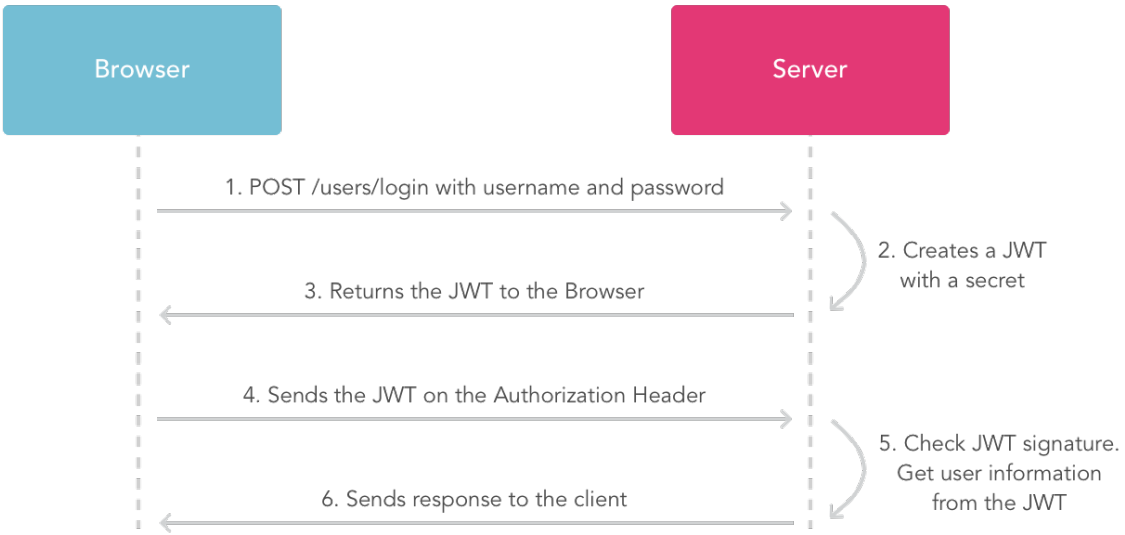

FIGURA 2.4: Funcionamento de JWT [5]

2.3.3 *AngularJS*

AngularJS é uma *framework* estrutural para aplicações *web* dinâmicas. Foi desenvolvida em 2009 e é atualmente mantida pela *Google*. Esta é *open-source* e permite estender a sintaxe do HTML para expressar componentes da aplicação de forma clara e sucinta. Em *AngularJS*, com dados de ligação (*data binding*) e com a injeção de dependências, reduz-se à quantidade de código que se teria de escrever.

AngularJS tenta minimizar a diferença entre um documento HTML *centric* e o que uma aplicação precisa. Introduz novas construções HTML. *AngularJS* ensina ao *browser* uma nova sintaxe, por meio de uma nova construção a que se chama de diretiva [2].

AngularJS permite o desenvolvimento de *Rich Internet Applications* (RIA). *AngularJS* fornece opções a quem desenvolve aplicações do lado cliente e possibilita a adoção de desenvolvimento baseado em MVC. Aplicações desenvolvidas com *AngularJS* são compatíveis com a generalidade dos *browsers* recentes já que este adequa o código *JavaScript* para cada *browser*. *AngularJS* foi desenvolvido para permitir a criação de aplicações *Create, Read, Update and Delete* (CRUD). Este tipo de aplicações representa a maioria das aplicações *web*[2].

Em suma, o *AngularJS* facilita a criação de aplicações de uma única página, fornecendo a ligação dos dados ao HTML, dando assim ao utilizador uma experiência rica e responsiva. *AngularJS* fornece componentes reutilizáveis [56].

2.3.4 *Satellizer*

O *Satellizer* é vocacionado para E2E. É um módulo de autenticação de domínios externos baseado em *tokens* para *AngularJS*. *Satellizer* contém suporte para a autenticação com contas *Google*, *Facebook*, *LinkedIn*, *Twitter*, *Instagram*, *GitHub*, *Bitbucket*, *Yahoo*, *Twitch*, *Microsoft*

(*Windows Live*) *OAuth providers*, bem como suporte para autenticação tradicional ou por *email* [63]. Pode ser adicionado qualquer domínio de autenticação que suporte *OAuth 1.0* [61] ou *OAuth 2.0* [62].

2.3.5 *Ionic*

O *Ionic* é um *SDK open-source* que possibilita o desenvolvimento de aplicações móveis híbridas. O *Ionic* foi criado em 2013 por Max Lynch, Ben Sperry e Adam Bradley da empresa *Drifty Co* [24]. Foi construído sobre *AngularJS* e *Apache Cordova*. As aplicações desenvolvidas em *Ionic* assentam em tecnologias *web* como o HTML5, o CSS, o *JavaScript*.

Estas aplicações consistem essencialmente em pequenos sites que contém também acesso à camada nativa da plataforma *Android* ou *IOS*. As aplicações híbridas têm muitas vantagens sobre as aplicações nativas, especificamente em termos de suporte à plataforma e em termos de velocidade de desenvolvimento [24]. *Ionic* integra suporte para *Android* 4+ e *IOS* 6+.

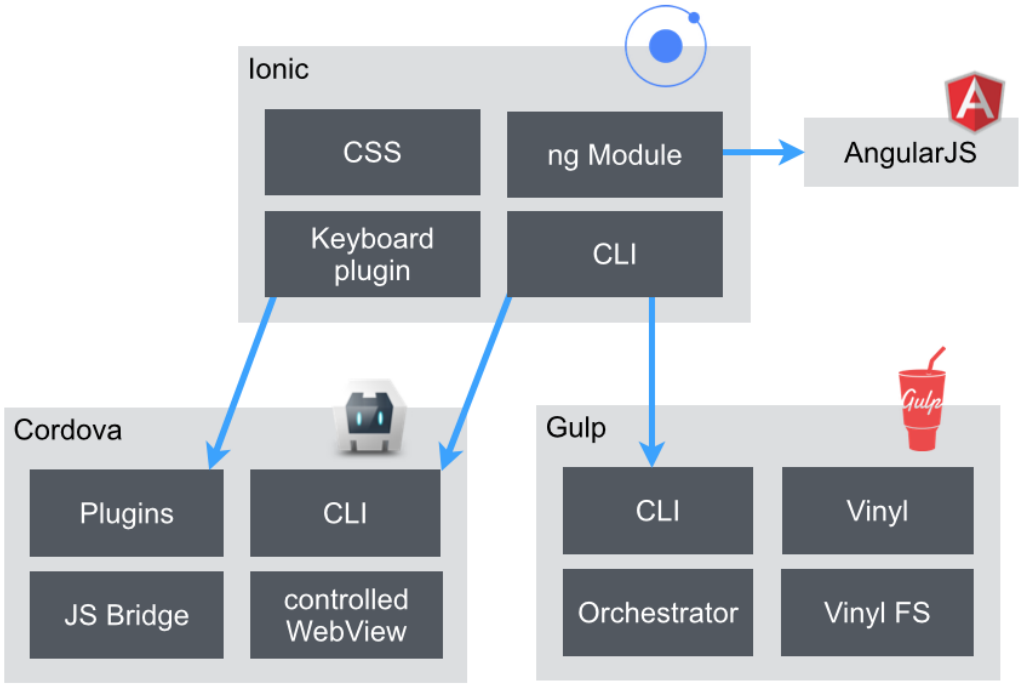

FIGURA 2.5: Arquitetura da *framework Ionic* [35]

Como se constata na Figura 2.5, o *Ionic*, no seu *core*, é constituída por quatro partes: CSS, *ng module*, *keyboard plugin* e *CLI*. O CSS define um *layout* de base otimizado para *smartphones* e que serve de base para a aplicação. O *ng Module* representa o *AngularJS* e define todas as diretivas, que incluem *navigation patterns* e outras, de modo a não despender muito tempo na sua especificação. O *Keyboard plugin* providencia um conjunto acrescentado de informações sobre o estado atual da aplicação. Por último, o *CLI* permite agir como uma espécie de *proxy* para o *Cordova* e para o *Gulp CLI* [35].

O *Ionic* recorre a uma arquitetura MVC com *AngularJS* para possibilitar a construção de aplicações otimizadas para *smartphones*. Integra e estende componentes CSS, dotando-os de

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.7%
4.1.x	Jelly Bean	16	9.0%
4.2.x		17	12.2%
4.3		18	3.5%
4.4	KitKat	19	36.1%
5.0	Lollipop	21	16.9%
5.1		22	15.7%
6.0	Marshmallow	23	0.7%

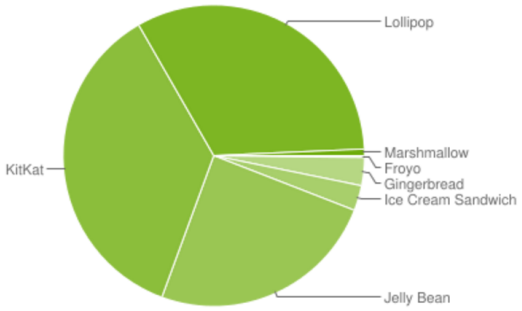

FIGURA 2.6: Tabela com as diferentes versões que *Ionic* suporta em *Android* [27]

funcionalidades, com *JavaScript*, para suprimir insuficiências dos elementos móveis do HTML. O *Apache Cordova* fornece *plugins* e a API necessária para utilizar as funções nativas do *smartphone* com código *JavaScript*. O *Ionic CLI* possibilita a gestão de aplicações, incluindo funções para iniciar, construir, executar e emular aplicações em *Ionic*.

Para iniciar com a *framework Ionic*, são fundamentais para a sua instalação quatro componentes de *Software* [57]: o *NodeJS*, o *Android SDK*, o *XCode* e *Cordova e Ionic*.

O *NodeJS*, é a plataforma de base necessária para criar aplicações móveis utilizando *Ionic*. Necessário a instalação de *NPM*. Em relação ao desenvolvimento, se este for na plataforma *Windows* e as aplicações a desenvolver forem para plataforma *Android*, então este, deve conter a configuração do *Android SDK* na máquina. Agora, se o desenvolvimento for na plataforma *Mac* e as aplicações a desenvolver forem para a plataforma *IOS*, então este, deve conter a configuração do *Xcode* na máquina. A máquina deve conter do mesmo modo, os principais *SDKs* necessários para começar a desenvolver com *Ionic*, que são como mencionados *Cordova e Ionic* [57].

O *Ionic* apenas permite desenvolver aplicações para versões 4.4 e superiores do *Android*. Tal não é considerado como uma limitação já que a generalidade dos equipamentos têm esta versão ou superior (69,4%), como se pode ver pela Figura 2.6.

Ionic permite o desenvolvimento de aplicações híbridas, ou seja, o utilizador pode efetuar o *package* das aplicações para *IOS*, *Android*, *Windows Phone* e *Firefox OS*. Tal economiza uma grande quantidade de tempo e trabalho. As aplicações *Ionic* são construídas de forma modular, o que as torna fáceis de manter e atualizar. Como limitação do *Ionic*, pode ser complexo combinar funcionalidades nativas numa mesma aplicação [58].

2.4 Conclusão

As tecnologias de comunicação disponíveis hoje em dia na generalidade dos *smartphones* são variadas, cada uma com um propósito particular. NFC e o *Bluetooth* são duas tecnologias de comunicação em proximidade com relevo. O desenvolvimento aplicacional para a *web* e *smartphones* está cada vez mais facilitado com a eclosão de plataformas de desenvolvimento, ou *frameworks*, que possibilitam o desenvolvimento rápido e multi-plataforma. Algumas para desenvolvimento mais genérico, como é exemplo o *Ionic* ou *AngularJS*, outras mais específico como são exemplo o *JSON Web Token* e o *Satellizer*. Estas *frameworks* são construídas sobre linguagens já existentes de que são exemplo o HTML5 e o *Javascript*.

Capítulo 3

Trabalho Relacionado

O interesse por formas seguras de enviar mensagens *online* cresceu substancialmente. Agora, mais do que nunca, procuram-se formas mais eficazes de cifrar a informação que enviamos uns aos outros. Neste capítulo, são descritas algumas aplicações de troca de mensagens com suporte para a confidencialidade da informação trocada, que estão disponíveis na *Google Play*, *App Store*, entre outros locais para instalação. Este trabalho baseou-se em parte no trabalho da EFF, uma organização sem fins lucrativos que defende a liberdade civil no mundo digital, sobre a segurança das aplicações de conversação atualmente existentes. A EFF defende a existência de aplicações de conversação fortemente seguras e ao mesmo tempo simples de usar. É apresentada uma comparação entre as várias aplicações.

3.1 Protocolos criptográficos de suporte

Várias aplicações de conversação tem como objetivo garantir a confidencialidade das mensagens trocadas entre os seus utilizadores. Para tal, surgiram vários protocolos que permitem que tal confidencialidade seja assegurada. Alguns, com garantias adicionais como a confidencialidade futura. O *Off-the-Record* (OTR) é um dos primeiros exemplos. Este foi sendo evoluído, tendo dado lugar primeiramente ao *Silent Circle Instant Message Protocol* (SCIMP) e, mais tarde, ao *Double Ratchet*. Estes mecanismos, por serem disponibilizados em contexto de código aberto, acabaram por ser utilizados por várias aplicações de conversação atualmente disponíveis.

OTR

Uma das propriedades críticas de segurança OTR é o *forward secrecy*[1]. Esta é uma caraterística fundamental de qualquer protocolo que privilegia a confidencialidade. Para isso, OTR faz uso de chaves efémeras para cada sessão de troca de mensagens, sendo que estas permanecem em memória apenas por um curto período de tempo [45].

[1] *Forward secrecy* representa a capacidade de um sistema manter a confidencialidade de mensagens passadas mesmo com a divulgação de chaves criptográficas atuais/futuras.

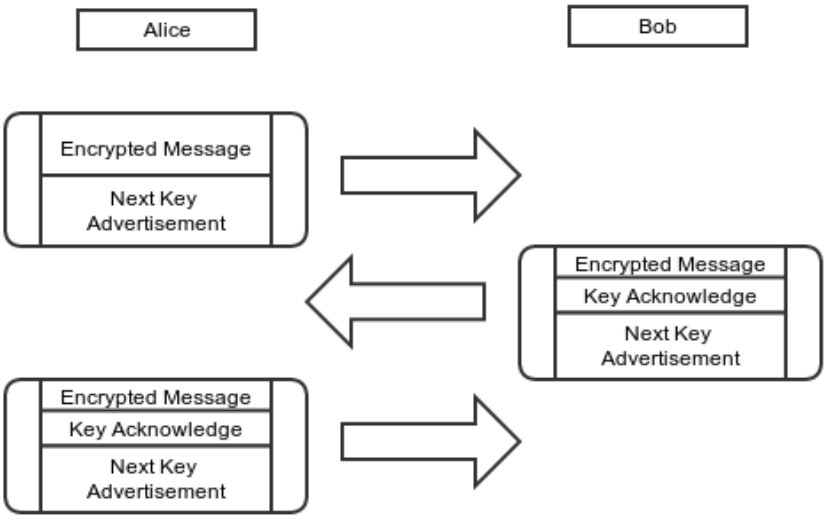

FIGURA 3.1: OTR *ratchet* "*three step ratchet*"[45]

O funcionamento do OTR *ratchet*, que assenta num algoritmo a três passos, pode ser visto na Figura 3.1. O seu funcionamento pode ser resumido, numa hipotética troca de mensagens entre *Alice* e *Bob*, da seguinte forma:

1. *Alice* envia uma mensagem cifrada para *Bob*. A nova chave *Diffie-Hellman*, a utilizar por *Alice* no futuro, é anexa à mensagem.
2. *Bob* envia uma mensagem cifrada para *Alice*. Anexa-lhe a sua futura chave *Diffie-Hellman* e o reconhecimento da sua nova chave anunciada por *Alice*.
3. *Alice* anuncia que reconheceu a chave da próxima vez que enviar uma mensagem.

O OTR foi pensado originalmente para aplicações de mensagens instantâneas. Nas aplicações, depois de estabelecida uma sessão, os utilizadores podem enviar mensagens de forma segura entre si, porque a troca de chaves efémeras *Diffie-Hellman* no início de cada sessão seria provavelmente suficiente.

Com a utilização de sessões de longa duração, o protocolo OTR deixa algo a desejar. Se um emissor transmite algo para um recetor, e o recetor não responder por alguns dias, o emissor tem de manter o material criptográfico utilizado para cifrar a mensagem durante vários dias. Para colmatar este obstáculo, surgiu o SCIMP [45].

SCIMP

O SCIMP utiliza uma chave criptográfica por mensagem, em que cada chave é derivada, recorrendo a uma função criptográfica de resumo (*hash*), da última chave utilizada. O seu funcionamento pode ser resumido, numa hipotética troca de mensagens entre *Alice* e *Bob*, da seguinte forma:

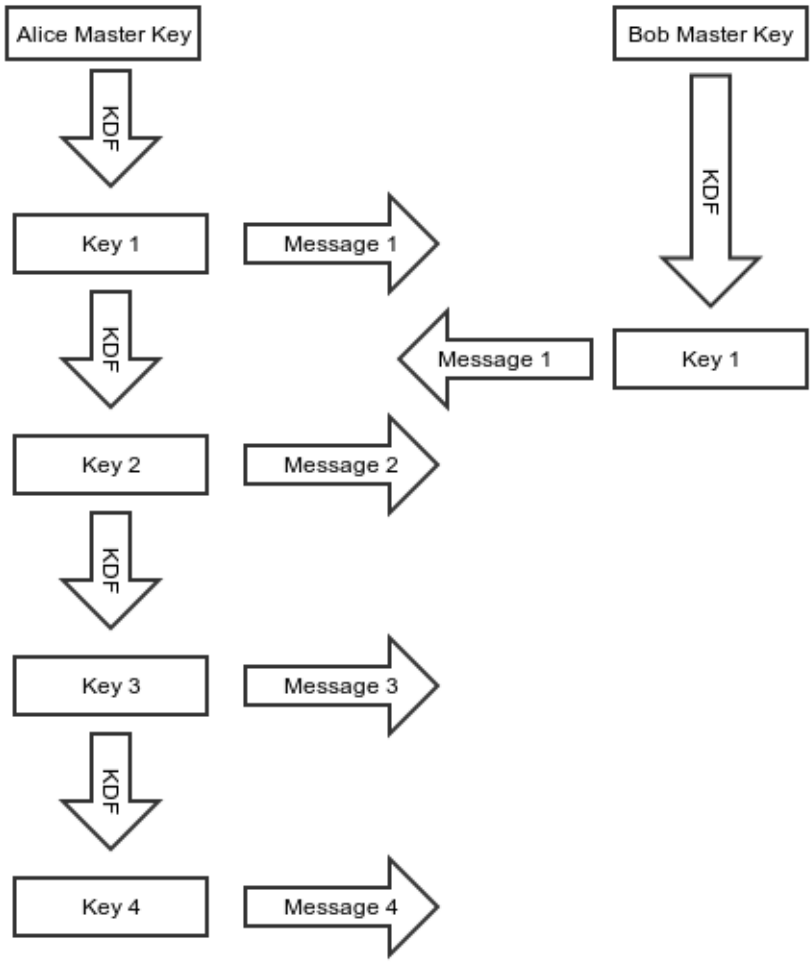

FIGURA 3.2: Geração de chaves no SCIMP [45]

1. *Alice*, assim que envia uma mensagem para *Bob*, calcula a nova chave aplicando uma *Key Derivation Function* (KDF) sobre a chave atual. A KDF em uso pode ser resumida como uma função de *hash*.

2. *Alice* destrói imediatamente a sua atual chave de cifra, substituindo-a pelo resultado da KDF.

O protocolo SCIMP contém excelentes propriedades de *forward secrecy*, porque cada mensagem é cifrada com uma chave diferente, mas sofre do problema de, com a divulgação de uma chave, as chaves futuras ficam também comprometidas. Já o protocolo OTR não sofre deste problema, porque chaves futuras não são derivadas de chaves passadas, mas usa cada chave para cifrar mais do que uma mensagem.

A junção das propriedades de cada um destes protocolos tendo em vista a redução do número de interações resultou num *Two-step DH* ratchet (ver Figura 3.3). Sendo que, neste caso todo o processo fica mais simplificado, e a mensagem com formato mais simples. O seu funcionamento pode ser resumido, numa hipotética troca de mensagens entre *Alice* e *Bob*, da seguinte forma:

1. *Alice* gera uma nova chave efémera ECDH A1 e utiliza para cifrar as mensagens a enviar.

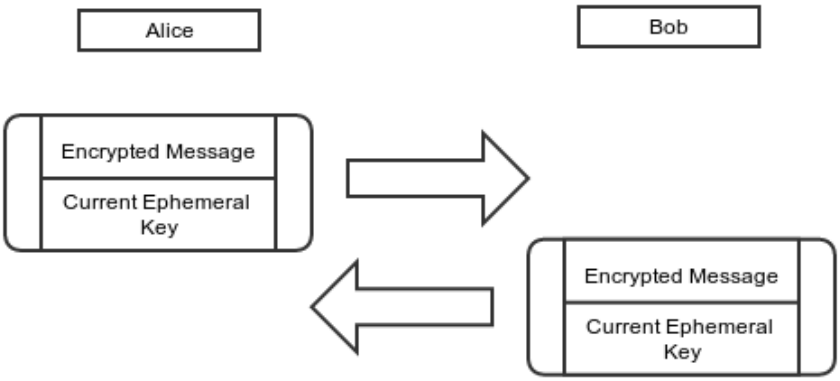

FIGURA 3.3: *Two-step DH* ratchet [45]

2. *Alice* recebe uma mensagem com a nova chave efêmera ECDH de *Bob* B1. Em seguida, destrói A1 e gera A2 que irá utilizar para cifrar a mensagem seguinte.

Algoritmo *Double Ratchet*

O *Double Ratchet* pode ser visto como uma evolução conjunta do OTR e do SCIMP. A Figura 3.4 apresenta uma perspetiva do seu funcionamento, tendo início pela *Alice*.

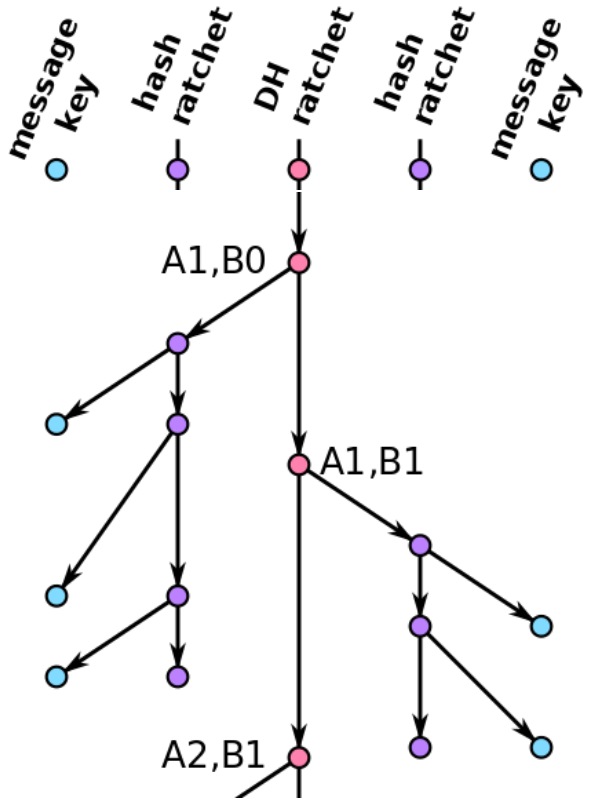

FIGURA 3.4: Diagrama do princípio de funcionamento [60]

Cada DH *ratchet* é combinado com uma *Root Key* (RK) (derivando uma nova RK), bem como com uma *hash ratchet*[2] para gerar uma nova DH *ratchet*. Quando o recetor, neste caso o *Bob*, recebe uma mensagem cifrada, gera e envia uma chave pública efémera Curve25519. Utilizando a chave efémera, juntamente com a sua própria chave (a chave que ele também envia), consegue calcular a chave corrente, e a respetiva RK, para decifrar a mensagem. Logo que é estabelecido o nível anterior, é gerada uma nova DH *ratchet* e assim sucessivamente [45].

3.2 *TextSecure*

TextSecure é uma aplicação móvel livre e *open-source* para a plataforma *Android* que permite o envio de mensagens de texto cifradas. Foi lançada pela primeira vez em 2010. É considerada como uma das aplicações mais usuais para o envio de mensagens seguras. A aplicação *TextSecure* foi desenvolvida pela *Open Whisper Systems*, que afirma oferecer suporte à encriptação E2E [46]. Após aquisição do *WhatsApp* pelo *Facebook*, o *TextSecure* ganhou notoriedade entre grupos de utilizadores promotores da privacidade e conta com mais de 500.000 instalações através da *Google Play*.

Durante a utilização normal do *TextSecure* no seu *smartphone*, o conteúdo ou texto das mensagens armazenadas pela aplicação é cifrado localmente pela aplicação, i.e. em caso de acesso físico ao equipamento por parte de um atacante, ele não será capaz de obter os dados. Algumas das operações de cifra são implementadas usando a biblioteca nativa do *Android*.

O protocolo inicialmente utilizado pelo *TextSecure* era um protocolo derivado do OTR. Este protocolo compreendia as seguintes fases: o registo, envio/receção de uma primeira mensagem, o envio de uma mensagem *follow-up*, e o envio de uma resposta [54]. O *push* do canal de dados pela aplicação *TextSecure* recorre a um servidor da *Whispersync* que permite a retransmissão das mensagens para o destinatário. As partes comunicam com o servidor *TextSecure* via REST-API com *Hypertext Transfer Protocol Secure* (HTTPS). O certificado do servidor *TextSecure* é auto-assinado, e a assinatura do certificado CA é embutido no *TextSecure*. A entrega da mensagem real é executada via *Google Cloud Messaging* (GCM) que, basicamente atua como um intermediário de entrega de mensagens [54].

Atualmente, a *Open Whisper Systems* adota uma estratégia diferente para garantir a confidencialidade das mensagens. O *TextSecure* agora implementa o algoritmo *Double Ratchet*[3].

3.3 *Signal*

Signal é o sucessor de *RedPhone* que é uma aplicação de chamadas de voz cifradas, e de *TextSecure*, aplicação móvel para envio de mensagens de texto cifradas. As versões *beta* de *RedPhone* e *TextSecure* foram lançadas pela *WhisperSystems* [30].

[2] É uma *hash* que é iterada a cada mensagem enviada/recebida
[3] Também conhecido como *Axolotl ratchet*

Signal foi lançada em 2015 e é uma aplicação de cifra de mensagens instantâneas e chamadas de voz para as plataformas *Android* e *IOS*. Esta aplicação utiliza cifra E2E para garantir a confidencialidade de todas as comunicações entre utilizadores da mesma aplicação. *Signal* pode ser utilizada para envio e receção de mensagens instantâneas cifradas, mensagens de grupo, anexos e conteúdos multimédia.

As chaves que são utilizadas para cifrar as comunicações do utilizador são geradas e armazenadas nos extremos comunicantes (ou seja, nos *smartphones* dos utilizadores e não nos servidores) [12]. *Signal* foi construída com mecanismos para resistir a ataques MiTM. No caso das chamadas telefónicas, a aplicação *Signal* exibe uma palavra no ecrã, se as duas palavras corresponderem em ambas as extremidades da chamada, então a chamada é segura [48, 51]. Existe um mecanismo similar para as mensagens que consiste na verificação mútua de assinaturas digitais.

O protocolo utilizado pela aplicação *Signal* é *open source* e é conhecido como protocolo *Axolotl*. É um protocolo de cifra forte para sistemas de mensagens assíncronas. As bibliotecas utilizadas pela aplicação *Signal* estão também disponíveis para serem usadas por outras aplicações [30].

3.4 *Telegram*

Telegram foi lançado em 2013. É um serviço de mensagens instantâneas com principal foco na disponibilidade multi-plataforma. *Telegram* fornece cifra de mensagens E2E e mensagens *self-destructing*. A aplicação oferece dois tipos de *chats*: *chat* padrão que usa cifra cliente-servidor e pode ser acedido a partir de vários *smartphones*; e o *secret chat (Telegram (Secret Chats))* que utiliza cifra E2E, onde só os dois dispositivos que estão a comunicar entre si tem acesso. Do lado cliente o código é *open source*, enquanto que do lado do servidor é *closed-source* [50].

Em relação ao protocolo, é projetado para aceder a uma API no servidor para aplicações executadas em *smartphones*. O protocolo utilizado, de nome *MTProto*, usa cifra RSA2048 [6], cifra simétrica com AES256 bits e o protocolo de troca de chaves *Diffie–Hellman* [11, 34]. O protocolo *MTProto* é subdividido em três componentes independentes [49]: componente de alto nível, componente de cifra e componente de transporte. O componente de alto nível define o método pela qual as consultas e as respostas da API são convertidas em mensagens binárias. A componente de cifra (autorização) define o método através da qual as mensagens são cifradas posteriormente a serem transmitidas. O componente de transporte define o método a utilizar por clientes e servidor para transmissão das mensagens sobre um protocolo de rede. Permite a entrega de *containers* cifrados em conjunto com um *header* externo, do cliente para o servidor e vice versa. Existem vários modos de transporte, sendo estes: HTTP, HTTPS, *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP).

As mensagens de texto, quando cifradas no *MTProto*, contêm sempre os seguintes dados: *salt* do servidor (64 bits), *ID* de sessão, número de sequência da mensagem, tamanho da mensagem e tempo.

Estes dados, por poderem ser verificados, tornam o sistema num sistema robusto. O cliente e o servidor trocam mensagens dentro de uma sessão. A sessão está ligada ao *smartphone* cliente, em vez de estar associada a uma conexão HTTP/HTTPS específica. Além disso, cada sessão está associada a uma chave de identificação de utilizador, através da qual a autorização é efetivamente realizada [49].

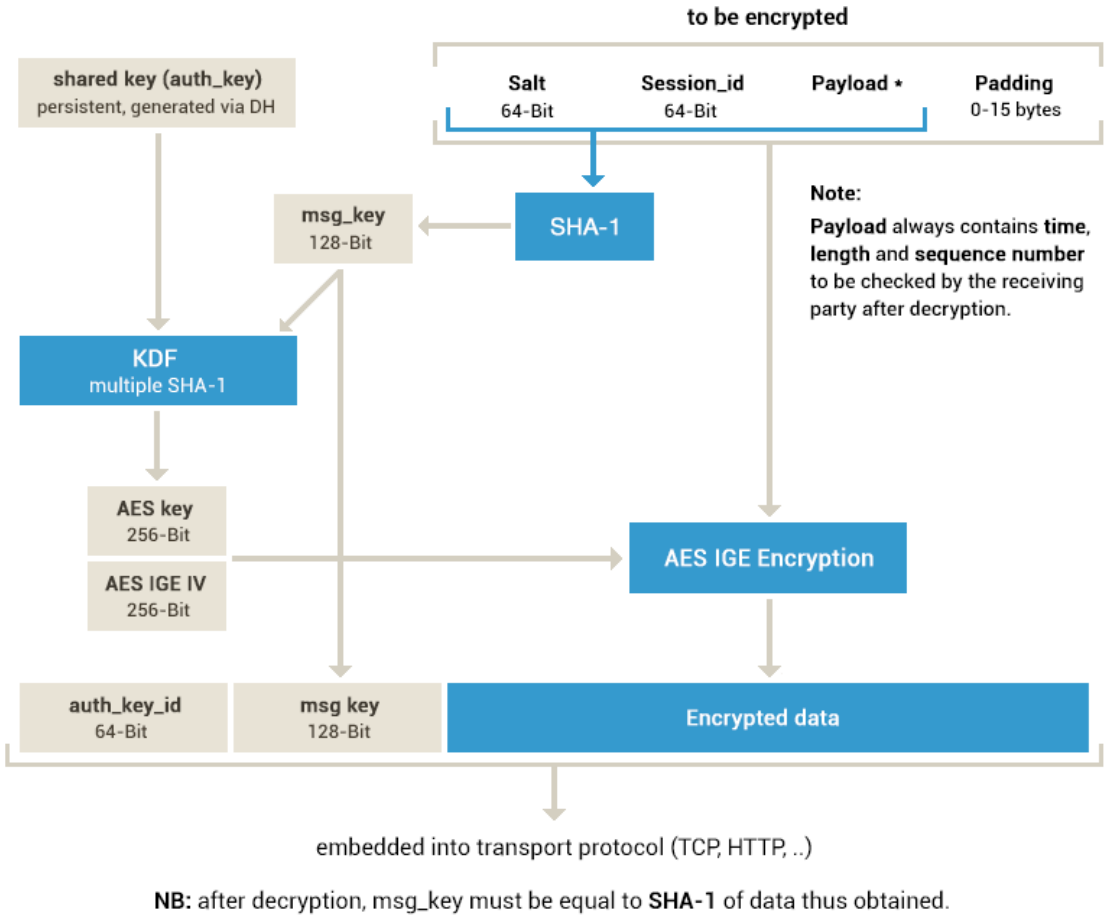

FIGURA 3.5: Protocolo *MTProto* (cifra cliente/servidor) [49]

Na Figura 3.5 está representado o protocolo *MTProto*. As mensagens são cifradas antes de serem enviadas pela rede. O cabeçalho externo, adicionado na parte superior da mensagem HTTP, é composto por: um identificador de chave com 64 bits (identifica uma chave de autorização para o par servidor/utilizador) e uma chave com 128 bits. A chave de utilizador juntamente com a chave da mensagem são utilizados para gerar uma chave de 256 bits e um vetor de inicialização de 256 bits, mais tarde utilizados para cifrar a mensagem usando AES256 com *Infinite Garble Extension* (IGE). A parte inicial da mensagem contém dados (sessão, *ID* da mensagem, número de sequência e o *salt* do servidor) que influenciam a chave da mensagem. A chave da mensagem é calculada usando a função de resumo SHA-1 sobre o corpo da mensagem. As mensagens com várias partes são cifradas como uma única mensagem [49].

3.5 *WhatsApp*

WhatsApp é uma das aplicações móveis mais utilizadas, que faz uso da *Internet* para envio de mensagens de texto e partilha de conteúdos (áudio, vídeo, entre outros). Foi comprada pelo *Facebook* em 2014. *WhatsApp* é uma das mais populares para *Android*, *IOS* e outros dispositivos móveis, tornando-se no maior serviço de troca de mensagens, contemplando um número base em utilizadores de até 900 milhões [43] (Setembro de 2015).

O mecanismo utilizado para trocar mensagens entre utilizadores é o de armazenar e transmitir. Ou seja, quando um utilizador pretende enviar uma nova mensagem, esta em primeiro lugar é armazenada no servidor *WhatsApp*, e em seguida, o servidor solicita repetidamente ao utilizador de destino que receba a mensagem. Recebida a mensagem por parte deste, deixa de estar disponível na base da dados do servidor (o servidor mantém por apenas 30 dias uma mensagem quando não entregue ao destinatário) [18].

O protocolo que o *WhatsApp* utiliza é o mesmo que o *TextSecure* (*Axolotl* ou algoritmo *Double Ratchet*). Anteriormente, o protocolo utilizado pela *WhatsApp* era uma versão baseada no *Extensible Messaging and Presence Protocol* (XMPP) [36], de nome *FunXMPP*. O *FunXMPP* permitia reduzir o número de *bytes* utilizados na troca de mensagens. Como é uma aplicação orientada para dispositivos móveis, que muitas vezes não possuem boa ligação à Internet, quanto menor carga, melhor [59].

Existem dois tipos de autenticação que o utilizador pode usar quando faz a ligação aos servidores: o *full handshake* e o *half handshake* [14]. O *full handshake* é executado quando o utilizador se liga pela primeira vez ao servidor e consiste em três mensagens. A mensagem inicial *auth* é enviada pelo utilizador para o servidor, não cifrada, e contém o número de utilizador e o método de autenticação. O servidor retorna uma mensagem do tipo *challenge*, utilizada para gerar a chave de sessão com *PasswordBased Key Derivation Function 2* (PBKDF2). Num passo posterior, o utilizador cria uma mensagem *response* que é cifrada com a chave de sessão gerada. Por último, o servidor responde com *success* ou *failure* se a autenticação for bem sucedida ou não, respetivamente. O *Half handshake* consiste no envio da mensagem *auth*, que já contém os dados da mensagem *response*, servindo-se da sessão anteriormente estabelecida.

3.6 *Threema*

Threema é uma aplicação livre e *open source* para troca de mensagens instantâneas cifradas que opera em sistemas *IOS*, *Android* e *Windows Phone*. Suporta cifra E2E de todas as suas comunicações, incluindo mensagens de texto e de voz, *chats* de grupo, partilha de ficheiros e até mesmo mensagens de estado [52]. O nome *Threema* representa *End-to-End Encrypting Messaging Application* (EEEMA). *Threema* foi fundada em 2012, sendo desenvolvida na Suíça pela empresa *Threema GmbH*, e em Junho de 2015, *Threema* alcançou os 3,5 milhões de utilizadores, a maioria deles de países de língua alemã.

É uma aplicação que permite anonimato completo do utilizador. A cada utilizador, no momento em que inicia o uso da aplicação, é atribuído de forma aleatória um *Threema ID* para identificação. Não é necessário assim um número de telemóvel ou endereço *email*, permitindo um anonimato completo.

A sincronização com *Threema* é possível através de números de telefone ou *email*, se o utilizador permitir que a aplicação sincronize com a sua lista de contactos. Os utilizadores podem verificar a identidade dos seus contactos *Threema* digitalizando o seu QR *code*, quando se encontram fisicamente. O QR *code* contém a chave pública do utilizador respetivo. Utilizando este recurso, os utilizadores podem ter a certeza de que eles contêm a chave pública correta dos utilizadores das suas conversações (fornece segurança num ataque MiTM) [52].

Threema foi projetada para gerar o mínimo de dados nos servidores quanto possível: lista de contactos são geridas apenas no *smartphone*, e as mensagens são excluídas imediatamente após terem sido entregues.

Threema usa cifra assimétrica para proteger as mensagens entre o emissor e o recetor, bem como a comunicação entre a aplicação e os servidores. *Threema* possui duas camadas diferentes de cifra para proteger as suas mensagens entre o emissor e o recetor (ver Figura 3.6) [53]: a camada de cifra E2E e a camada de transporte.

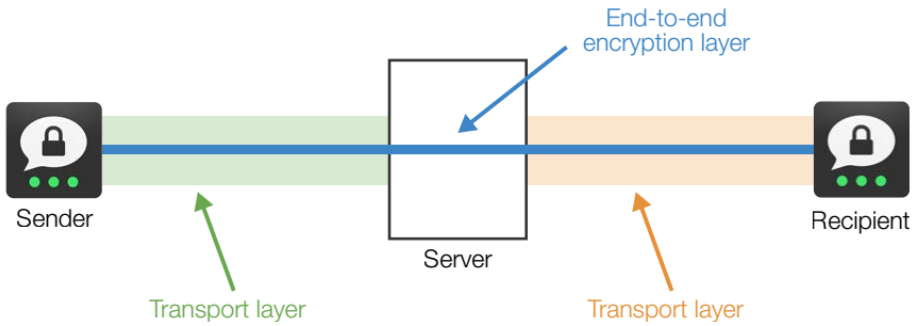

FIGURA 3.6: Camadas de cifra que *Threema* utiliza para proteger as mensagens [53]

A camada de cifra E2E é entre o emissor e o recetor. E, para cada mensagem cifrada E2E, é novamente cifrada no transporte entre o cliente e o servidor, a fim de proteger a informação contida nos cabeçalhos de transporte. A parte crucial é que a camada de cifra E2E passa através do servidor ininterrupta (o servidor não pode remover a camada de cifra interna).

O *Threema* usa a *Networking and Cryptography library* (NaCl) para cifra e autenticação de mensagens. Quando um utilizador se regista na aplicação *Threema* pela primeira vez, é executado o seguinte processo [53]:

1. A aplicação gera um novo par de chaves e armazena-o de forma segura no *smartphone*.
2. A aplicação de seguida envia a chave pública para o servidor.
3. O servidor armazena a chave pública e atribuí-lhe um novo *Threema ID*.

4. A aplicação armazena o *Threema ID* que recebeu, juntamente com a chave pública e privada de forma segura no *smartphone*.

A chave pública de cada utilizador é armazenada no servidor de diretório, juntamente com o seu *Threema ID*. Qualquer utilizador pode obter a chave pública para um determinado *Threema ID* consultando o servidor.

3.7 *Wickr*

Wickr é uma aplicação gratuita de mensagens que oferece confidencialidade, autenticação e usabilidade. Esta aplicação pode ser instalada em *Android*, *IOS*, *Windows*, *Mac* e *Linux* [22]. *Wickr* inclui a capacidade de definir um *time-to-live* para cada mensagem. A aplicação do recetor apaga a mensagem cifrada do *smartphone*, assegurando que a mensagem não possa ser recuperada.

O protocolo *Wickr Secure Messaging* foi projetado especificamente para impedir que os seus servidores possuam chaves ou informações detalhadas dos seus utilizadores. Suporta comunicações E2E seguras, implementando múltiplas camadas de cifra e autenticação [21]. Na Figura 3.7, encontram-se representadas as várias camadas de cifra utilizadas pela aplicação *Wickr* para proteção dos dados tanto em trânsito, como enquanto armazenados.

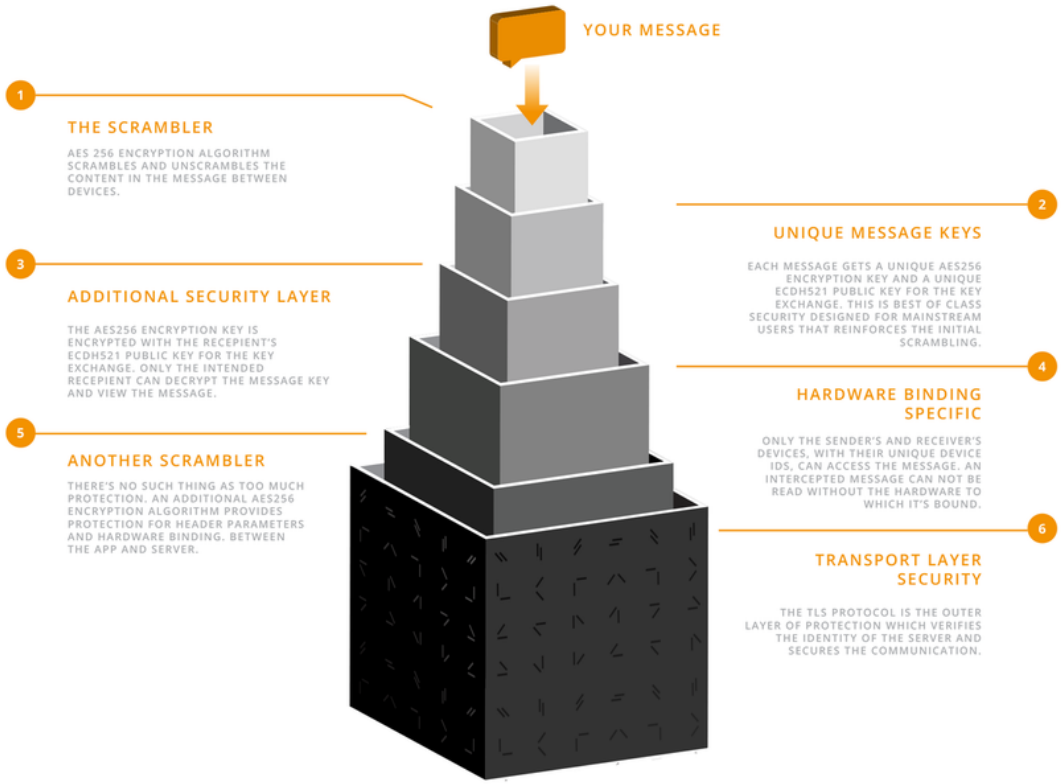

FIGURA 3.7: Camadas de cifra de proteção de dados da aplicação *Wickr* [21]

Os dados, armazenados e em trânsito, são cifrados com AES256. Cada mensagem é cifrada com uma nova chave de cifra, que é eliminada após a sua utilização (*Perfect Forward Secrecy*). As chaves de cifra das mensagens são cifradas com chave pública (ECDH521). Todo o conteúdo do utilizador é apagado depois deste terminar a sessão. O identificador único do *smartphone*, *Unique Device Identifier* (UDID), nunca é enviado para os servidores *Wickr* para manter o anonimato. O *Wickr's Secure Shredder* apaga todos os dados do *smartphone* de modo a não poderem ser recuperados. No final, todas as comunicações do utilizador são apagadas [21].

3.8 Comparação

A comparação das várias aplicações descritas neste capítulo está ostentada na tabela 3.1. Estas são abordadas segundo o *Secure Messaging Scorecard* da EFF [13]. O *Secure Messaging Scorecard* examina várias aplicações de conversação sob dois pontos de vista: segurança e usabilidade. Esta campanha está focada em aplicações de comunicação e inclui aplicações de *chat*, aplicações de mensagens de texto, aplicações de *email* e aplicações de vídeo chamada. As aplicações avaliadas têm uma grande base de utilizadores e, portanto, processam uma grande quantidade de comunicações confidenciais entre utilizadores individuais. O *Scorecard* da EFF pode ser visto como um estímulo para a inovação em torno de uma cifra forte das comunicações digitais. Em particular, foram analisados 5 fatores principais.

	Transporte	*Provider*	Entidades	Anonimato	Identidade
Text-Secure	**Sim**	**Sim**	**Sim**	**Não**	**Não**
Signal	**Sim**	**Sim**	**Sim**	**Sim**	**Não**
Telegram	**Sim**	**Não**	**Não**	**Não**	**Não**
Telegram (Secret-Chats)	**Sim**	**Sim**	**Sim**	**Não**	**Não**
WhatsApp	**Sim**	**Não**	**Não**	**Não**	**Não**
Threema	**Sim**	**Sim**	**Sim**	**Sim**	**Não**
Wickr	**Sim**	**Sim**	**Sim**	**Sim**	**Não**

Tabela 3.1: Comparação das *APP's* Relacionadas

O critério transporte requer que todas as mensagens ou comunicações sejam cifradas no caminho da comunicação, apenas exigindo a cifra do conteúdo da mensagem e não de nomes de utilizador ou endereços. O critério *provider* requer que todas as comunicações do utilizador sejam cifradas E2E, o que significa que as chaves necessárias para decifrar as mensagens devem ser geradas e armazenadas nos extremos comunicantes (nos *smartphones* dos utilizadores, e não nos servidores). O critério entidades requer que exista um método interno que permita aos utilizadores verificarem a identidade da entidade com quem estão a comunicar, bem com a integridade do canal, mesmo que os servidores ou terceiros sejam comprometidos [13]. O critério anonimato requer que a identidade do utilizador perante o servidor não seja conhecida. O critério

identidade requer que exista uma interação física prévia, como prova de identidade das pessoas intervenientes, entre os utilizadores para que estes possam trocar mensagens.

Por exemplo, a aplicação *TextSecure* permite que uma mensagem enviada por um utilizador circule cifrada durante o transporte até ao destinatário. A comunicação é cifrada E2E, evitando que o *provider* as possa ler. A identidade das entidades envolvidas é verificada. Não salvaguarda o anonimato dos seus utilizadores perante o servidor. Por último, o *TextSecure* não obriga a uma interação física prévia entre os utilizadores para que estes estabeleçam uma conversa.

Posto o título de exemplo, é de ressaltar que a aplicação *Threema* permite uma interação física entre os utilizadores. Esta interação é efetuada para verificar a identidade dos seus contactos *Threema* digitalizando o seu QR *code*. Isto é próximo do que se pretende com o requisito identidade, mas não satisfaz, já que não é necessário que aconteça antes de se dar a conversação.

3.9 Conclusão

Existem múltiplas aplicações de conversão para a generalidade dos *smartphones*. Preocupações com a confidencialidade, a integridade e o anonimato estão cada vez mais presentes. Tendo por base um trabalho anterior da EFF, foram analisadas várias aplicações segundo 5 fatores. Foi elaborada uma tabela comparativa. O fator identidade, que requer que os utilizadores interajam previamente e fisicamente como prova de identidade, não é suportado por nenhuma das aplicações analisadas.

Capítulo 4

EkoChat

O problema relacionado com o anonimato e a utilização de falsas identidades, ou falta de certeza de quem é o outro interveniente nas conversações, conduziu a que a principal motivação deste projeto procurasse uma solução para esta problema. Assim, propõem-se uma aplicação que permite a troca segura de mensagens entre utilizadores verificados previamente e com confidencialidade e anonimato perante o servidor.

4.1 Requisitos

A solução proposta tem como principais requisitos:

1. **Identidade:** A aplicação cliente deve ser capaz de garantir, aos intervenientes em cada conversão, a identidade dos mesmos.
2. **Confidencialidade:** A aplicação servidor não deverá ser capaz, em circunstância alguma, de aceder ao conteúdo das mensagens trocadas entre os utilizadores. As mensagens são cifradas E2E.
3. **Anonimato:** A aplicação cliente deverá garantir sempre o anonimato dos utilizadores perante o servidor.
4. **Usabilidade:** A aplicação cliente deverá ter um *interface* de utilização simples, intuitiva e amiga do utilizador.
5. **Portabilidade:** A aplicação cliente deverá correr nas plataformas móveis *Android* e *IOS*.

A análise efetuada a trabalhos relacionados permitiu concluir que nenhuma destas aplicações (ver Capítulo 3) cumpre com o requisito *identidade*, que requer uma interação física e prévia entre os utilizadores. Razão que motivou o presente trabalho, denominado de *EkoChat*.

4.2 Arquitetura da solução

A solução proposta visa principalmente a comunicação confidencial. A solução é implementada para todo o utilizador que disponha de um *smartphone*, com intenção de querer comunicar com outros de forma segura e sem que terceiros obtenham conhecimento de qualquer tipo de dados ou da informação trocada.

FIGURA 4.1: Troca, por NFC, do *ID_CHAT* entre utilizadores da aplicação

Todos os dados transmitidos na aplicação utilizam cifra E2E, para garantia de todas as comunicações com os outros utilizadores, possibilitando assim confidencialidade. Mesmo o servidor da aplicação não vai conter algum tipo de informação acerca dos utilizadores, preservando-se assim o anonimato dos mesmos.

Variável	Descrição
SK_i	**Chave simétrica de utilizador**
SSK_i	**Chave simétrica partilhada entre utilizadores e o servidor**
Id	**Identificador da conversação**
T_s	***Timestamp***
V	**Validade em número de dias (1, 10, 15, 30)**

TABELA 4.1: Estrutura de Dados *ID_CHAT*

Desta forma, de um modo elucidativo, para alcançar uma solução *zero knowledge* na troca de dados, associa-se um identificador a cada conversação estabelecida entre utilizadores. Cada conversação, como por exemplo entre dois utilizadores, contém um conjunto de troca de dados ou mensagens. A cada uma das conversações criadas, está associada a estrutura de dados apresentada na Tabela 4.1.

Os utilizadores, para poderem iniciar a troca de mensagens entre si, primeiramente têm de passar por uma interação física que consiste na troca de uma instância da estrutura de dados *ID_CHAT* por NFC ou *Bluetooth*. Com a troca do *ID_CHAT*, o utilizador passa a ter acesso

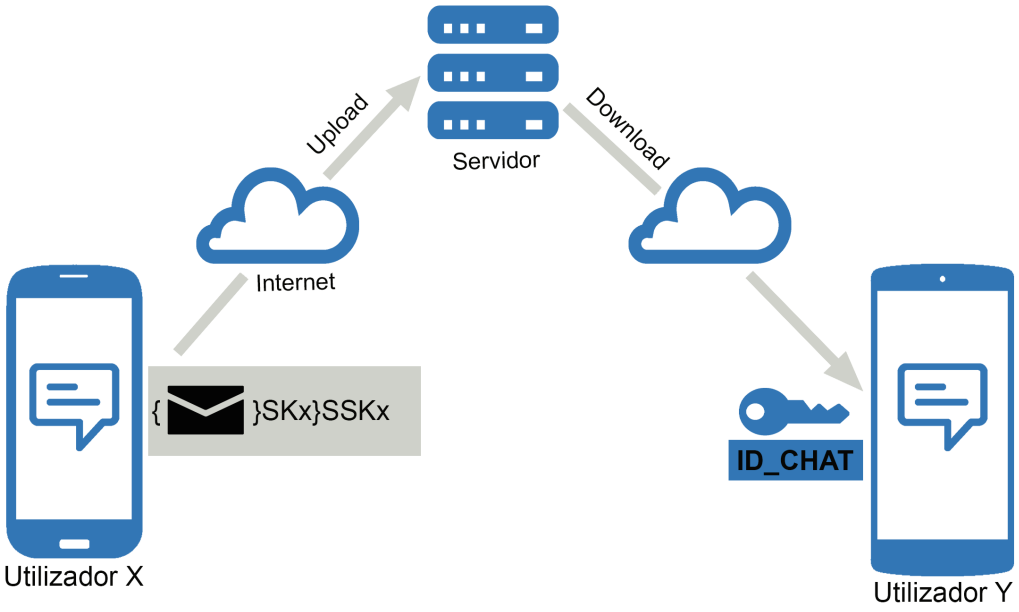

FIGURA 4.2: Arquitetura da aplicação

ao material criptográfico que lhe irá permitir ler ou escrever mensagens para com os restantes utilizadores, tal como representado na Figura 4.1.

A Figura 4.2 apresenta a arquitetura da solução proposta. Na representação, o anonimato dos utilizadores $\boldsymbol{x}$ e $\boldsymbol{y}$ é garantido através da estrutura de dados *ID_CHAT*. No envio de uma mensagem por parte do utilizador $\boldsymbol{x}$, a mensagem é cifrada de forma simétrica (com recurso a AES) garantindo a confidencialidade perante o servidor e perante terceiros: $\{\{Mensagem\}_{SK_x}\}_{SSK_x}$. Na receção da mensagem, o utilizador $\boldsymbol{y}$, por estar também na posse de *ID_CHAT*, pode decifrar a mensagem.

4.3 Perspetiva geral

A proposta de solução assenta no modelo cliente/servidor. A aplicação servidor será disponibilizada na Internet sob a forma de uma aplicação *web*. A aplicação cliente será uma aplicação para *smartphone* que faz uso da aplicação servidor. De seguida, são enumeradas as grandes funcionalidades que as aplicações a desenvolver deverão implementar.

A aplicação cliente começa com o *login*, onde é necessário que o utilizador introduza as suas credenciais de acesso (*username* e *password*). Se o *username* e *password* forem válidos, é lhe permite o acesso às demais funcionalidades. Após o *logoin*, é permitido que o utilizador envie e receba mensagens. Para criar uma nova conversa, o utilizador necessita de gerar uma estrutura de dados *ID_CHAT*, que representa a conversa, que é gerada e mantida unicamente nos extremos comunicantes (nos *smartphones*). Gerada a esturtura *ID_CHAT*, esta tem de ser partilhada com o(s) outro(s) utilizador(es). A partilha implica uma interação real entre os utilizadores já que se usam apenas tecnologias de comunicação por proximidade (NFC ou *Bluetooth*) nesta

partilha. Interação esta que evita falsas identidades ou a falta de certeza na identidade do outro interveniente nas conversações. Com esta partilha de *ID_CHAT*, ambos os utilizadores passam a possuir a capacidade de envio e receção de mensagens nessa conversa. Por último, o utilizador pode terminar a sessão, recorrendo à funcionalidade *logout*. A Figura 4.3 apresenta todas as funcionalidades alusivas ao utilizador.

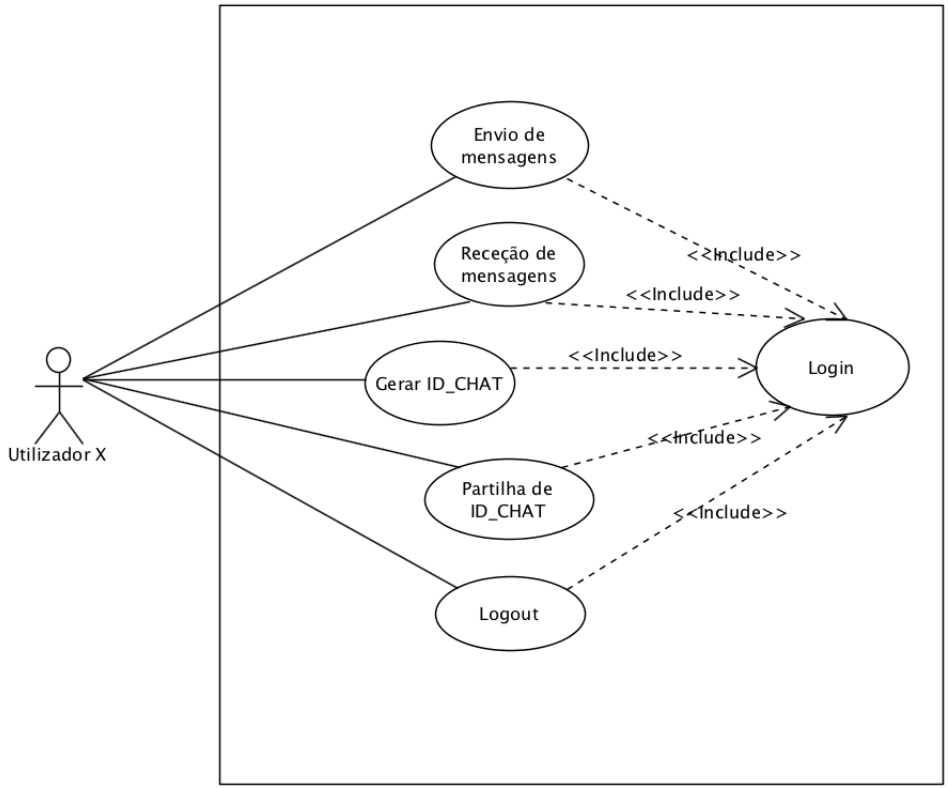

FIGURA 4.3: Esquema com funcionalidades associadas ao utilizador

A Figura 4.4 apresenta a sequência de mensagens, trocadas entre o utilizador $\boldsymbol{x}$ e o servidor, para que o utilizador obtenha a sua lista de mensagens. Para que se dê esta sequência de mensagens, o utilizador já tem de ter na sua posse o *ID_CHAT* dessa conversa.

Após validação das credenciais do utilizador (no *login*), este tem de obter uma chave criptográfica de sessão, ou *Token*. A obtenção deste *Token* é efetuada com um pedido *getToken* ao servidor (passo 1 da Figura 4.4). Este pedido contém a identificação do utilizador (*userID*), um *timestamp* ($userT_s$) e um bloco de dados cifrados com chave simétrica do utilizador $userSK_x$. O conteúdo cifrado contém novamente o *userID* e *timestamp*, bem como o *Token* anterior ($userT'_0$). Tanto o *timestamp* como o *userID* são enviados em *cleartext* como cifrados para permitir a sua validação por parte do servidor. No passo 1.1, em caso de sucesso, o servidor responde com uma nova chave de sessão ($userT''_0$) e com um *timestamp*. O *timestamp* é enviado em *cleartext* e cifrado para possibilitar a sua validação por parte do utilizador. Esta abordagem de validação por parte do servidor e cliente, repetindo campos das mensagens na parte cifrada e não cifrada, é adotada em todas as comunicações. Segue-se o pedido de lista mensagens com *getMessages* (ver passo 2). Este passo contém o *userID*, um *timestamp* e uma lista de identificadores de conversas ($chatID_1, chatID_2, ..., chatID_n$), subscritas pelo utilizador, cifradas com o *token* atual.

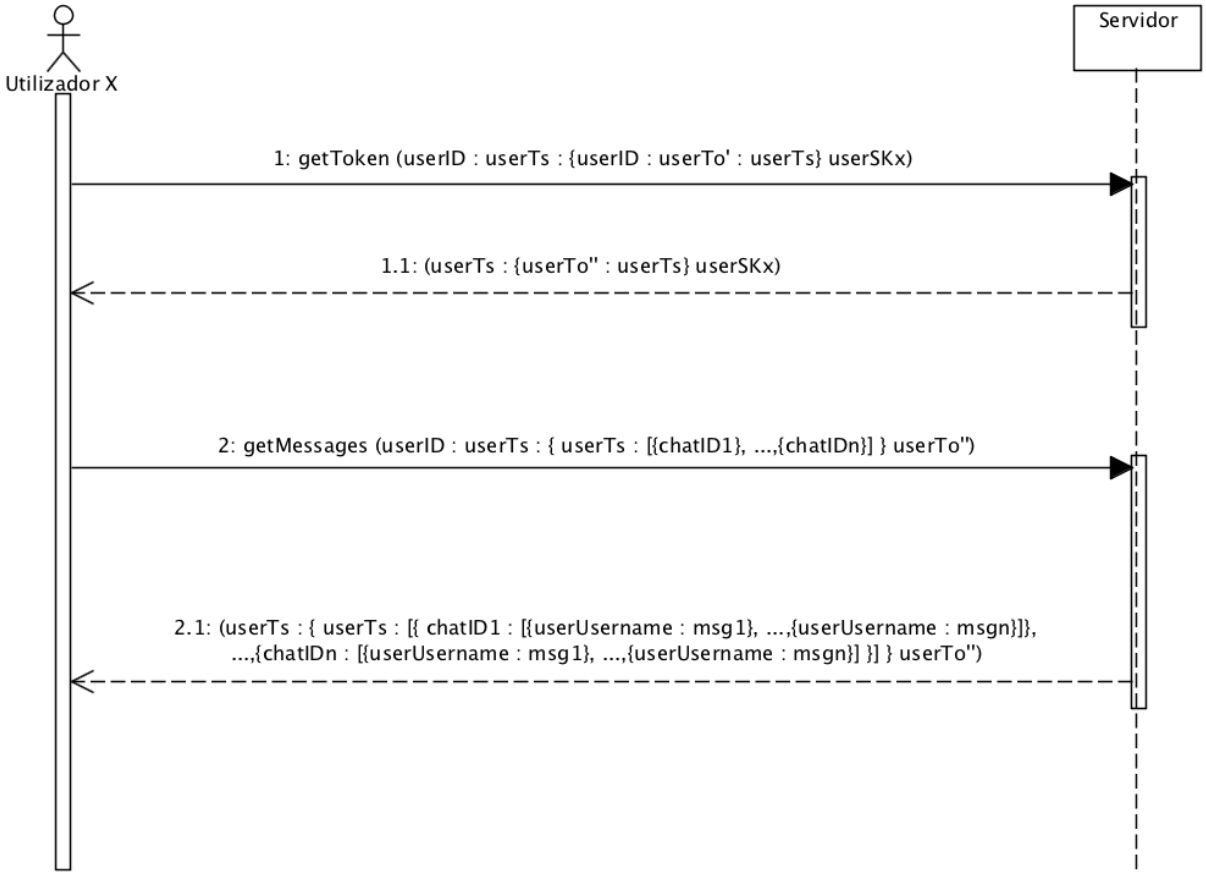

FIGURA 4.4: Esquema para obter lista de mensagens

A resposta, em caso de sucesso, é a lista de mensagens das várias conversas solicitadas (passo 2.1).

FIGURA 4.5: Esquema para gerar *ID_CHAT*

A Figura 4.5 apresenta um esquema de mensagens necessário para a criação de uma nova conversa e geração do respetivo *ID_CHAT*. Este passo requer que o utilizador esteja na posse de uma chave de sessão válida, i.e. tenha obtido um *token* numa operação de *login*. O objetivo deste passo é o de permitir ser o servidor a gerar o identificador (*chatID*) e a chave criptográfica ($chatSSK_x$) a utilizar em cada conversa. O pedido é implementado como uma operação *post* do HTTP (passo

1 da Figura 4.5) e contém o identificador do utilizador, um *timestamp* e um bloco cifrado. O bloco cifrado, além dos campos repetidos para validação, incluí a validade temporal para as mensagens da conversação. Esta validade é definida pelo utilizador e retrata o tempo máximo em que o servidor retém as mensagens, sendo eliminadas após este período. Em caso de sucesso, o servidor responde com o identificador da conversação *chatID* e com uma chave simétrica, partilhada entre o utilizador e o servidor, a utilizar para cifrar todas as mensagens enviadas para a conversa em questão. A chave criptográfica ($chatSSK_x$) tem como propósito principal dotar o servidor de um mecanismo para incluir na conversa apenas mensagens que realmente pertencem à conversa.

FIGURA 4.6: Esquema para envio de mensagem

A Figura 4.6 apresenta o esquema relativo ao envio de mensagens para uma conversa identificada pelo seu *chatID*. Este passo requer que o utilizador esteja na posse do *ID_CHAT* da conversa. O envio de uma mensagem para uma conversa é por meio de um *post* HTTP (passo 1 da Figura 4.6) que contém o identificador da conversa (*chatID*), um *timestamp* ($userT_s$) e um bloco cifrado com a chave criptográfica da conversa que é partilhada entre os utilizadores e o servidor ($chatSSK_x$). Este bloco cifrado conterá, por sua vez, os valores *chatID* e $userT_s$ para que o servidor possa confirmar a autenticidade da mensagem e proteger-se contra ataques de reenvio de mensagens (*replay atacks*). Inclui ainda um segundo bloco cifrado, desta vez cifrado com a chave criptográfica apenas partilhada entre os utilizadores da conversa ($chatSK_x$) que inclui a mensagem e o nome do utilizador que enviou a mensagem. A adoção do identificador do utilizador neste conteúdo permitiria que qualquer membro da conversa obtivesse este identificador e que tivesse a certeza da proveniência da mensagem. Tal foi considerado como facilitador de uma eventual quebra de anonimato, pelo que optou-se por recorrer a um *username* definido por cada utilizador para cada conversa.

4.3.1 Desenho da base de dados

O esquema lógico da base de dados apresentado na Figura 4.7 é o esquema a utilizar pelo servidor. O esquema da base de dados da aplicação é constituído por 5 entidades: *User*, *LanguagesUsers*, *Languages*, *ID_CHAT* e *Messages*.

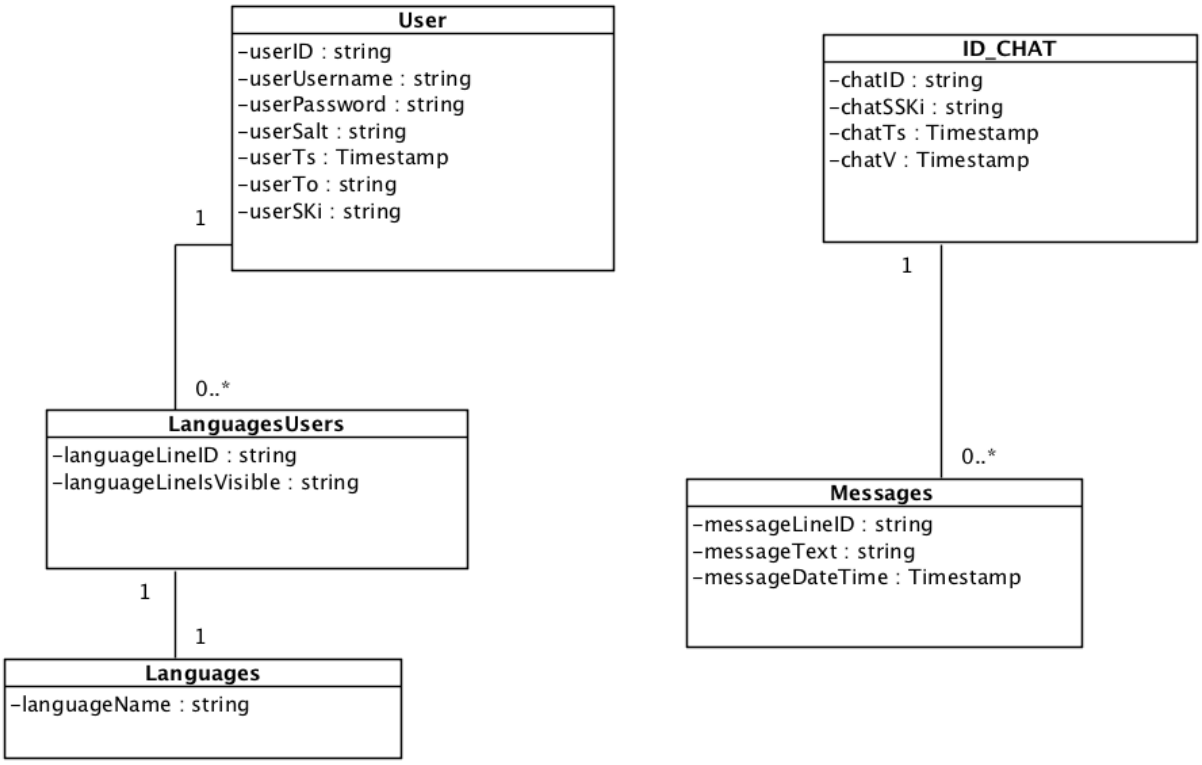

FIGURA 4.7: Esquema de classes do Servidor

A entidade *User* representa os utilizadores registados na aplicação. Esta entidade inclui o identificador do utilizador (um *email*), o *username*, a *password*, um *salt* (valor aleatório) para utilizar no processo de armazenamento da palavra passe, o último *timestamp*; a última chave de sessão ($userT_0$) e a chave simétrica do utilizador ($userSK_i$). A entidade *LanguagesUsers* retrata a informação sobre o idioma do utilizador. Esta entidade inclui um identificador para o idioma e uma indicação de visibilidade desse idioma (*languageLineIsVisible*). A entidade *Languages* definido o conjunto de idiomas disponíveis para o utilizador. A entidade *ID_CHAT* descreve cada uma das conversas existentes no servidor. Esta entidade inclui o identificador da conversação (*chatID*), a chave simétrica partilhada entre os utilizadores da conversa e servidor ($chatSSK_i$), o último *timestamp* e a validade das mensagens da conversação. A entidade *Messages* retrata as mensagens trocadas entre os utilizadores. Esta entidade inclui o identificador da mensagem, o texto da mensagem (*messageText*) e a data de envio da mensagem.

É de realçar que no esquema de dados, apresentado na Figura 4.7, não existe qualquer associação ou relacionamento entre as entidades *User* e *ID_CHAT*. A motivação para tal reside no facto de se querer uma total dissociação entre utilizadores e conversas para assim contribuir para a garantia de anonimato nas trocas de mensagens entre utilizadores. O esquema lógico da base de dados a adotar do lado do utilizador é globalmente similar ao do lado do servidor. A diferença reside no atributo ($chatSK_i$) da entidade *ID_CHAT*, que só deverá existir nos terminais dos utilizadores. Este atributo irá conter a chave simétrica de cifra de mensagens a utilizar pelos terminais, previamente ao seu envio para o servidor.

4.4 Desenvolvimento da solução

Para o desenvolvimento da solução as tecnologias usadas para implementar o servidor foi a *framework Laravel* e para implementar o cliente foi usada a *framework Ionic*.

Servidor *EkoChat - Laravel*

O servidor foi desenvolvido com recurso à *framework Laravel*. Esta *framework* é orientada para o desenvolvimento de aplicações *web* com suporte para aplicações *REST*. A Figura 4.8 apresenta a lista de pontos de acesso da aplicação que retratam as funcionalidades relevantes implementadas no servidor sob a forma de *Uniform Resource Locator* (URL)s. Estas funcionalidades, em *Laravel*, são designadas de *Routes* e indicam ainda o tipo de pedido HTTP suportado (*get, post*).

Method	URI	Name	Action
GET|HEAD	/		Closure
POST	api/generateChatroom		App\Http\Controllers\ChatRoomController@generateChatroom
POST	api/getChatRoomUpdates		App\Http\Controllers\ChatRoomController@getChatRoomMessages
POST	api/message	api.message.store	App\Http\Controllers\MessagesController@store
POST	api/register		App\Http\Controllers\Auth\AuthController@registerUser
GET|HEAD	api/signin	api.signin.index	App\Http\Controllers\UserAuthController@index
POST	api/signin		App\Http\Controllers\UserAuthController@signin
GET|HEAD	api/signin/user		App\Http\Controllers\UserAuthController@getAuthenticatedUser
GET|HEAD	home		App\Http\Controllers\HomeController@index
GET|HEAD	login		App\Http\Controllers\Auth\AuthController@showLoginForm
POST	login		App\Http\Controllers\Auth\AuthController@login
GET|HEAD	logout		App\Http\Controllers\Auth\AuthController@logout
POST	password/email		App\Http\Controllers\Auth\PasswordController@sendResetLinkEmail
POST	password/reset		App\Http\Controllers\Auth\PasswordController@reset
GET|HEAD	password/reset/{token?}		App\Http\Controllers\Auth\PasswordController@showResetForm
GET|HEAD	register		App\Http\Controllers\Auth\AuthController@showRegistrationForm
POST	register		App\Http\Controllers\Auth\AuthController@registerUser

FIGURA 4.8: Pontos de acesso da aplicação servidor

O ponto de acesso inicial é a *route api/signin* do controlador *UserAuthController*. Este ponto implementa o processo de autenticação de utilizadores numa função de nome *signin*. A função *signin* (Listagem 4.1) aceita como argumentos o *email* e *password* do utilizador. Se a autenticação do utilizador for válida é devolvido a chave de sessão ou *token*. Os *tokens* são gerados com recurso ao componente JWT.

```
(...)
    function signin(Request $request) {
    {
    // credentials from the request
    $credentials = $request->only('email', 'password');
    try {
        // attempt to verify the credentials and create a token for the user
        if (! $token = $this->auth->attempt($credentials)) {

            return response()->json(['error' => 'invalid_credentials '], 401);
            }} catch (JWTException $e) {
                return response()->json(['error' => 'could_not_create_token'], 500);
                }
            // return the token
            return response()->json(['token' => $token, 'user' => $this->auth->user()]);
        }
    }
(...)
```

Listagem 4.1: Autenticação do utilizador

A *route api/register*, do controlador *AuthController*, permite o registo de novos utilizadores (Listagem 4.2).

```
(...)
    protected function create(array $data) {
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => bcrypt($data['password']),
        ]);
    }
    public function registerUser(Request $request) {
        $validator = $this->validator($request->all());
        if ($validator->fails()) {
          return response()->json(['error' => true, 'msg' => $validator->messages()], 422);
        }
        $newUser = $this->create($request->all());
        return response()-> json(['user' => $newUser]);
      }
    }
(...)
```

Listagem 4.2: Registo do utilizador

Antes que se dê uma conversa entre dois utilizadores, um destes tem de gerar um novo *ID_CHAT* e, usando um mecanismo de comunicação por proximidade, comunica-lo ao outro utilizador. A Listagem 4.3 retrata o processo de criação de um novo *ID_CHAT*, que corresponde à *route*

api/api/generateChatroom do controlador *ChatRoomController*. O pedido de criação de um novo *ID_CHAT* é efetuada via *post* HTTP para o servidor. O pedido inclui o *userID*, a validade da conversação e um *timestamp*. A função *generateChatroom* devolve uma estrutura em formato JSON que contém um identificador único, uma validade temporal e uma chave simétrica AES ($chatSSK_i$) partilhada entre o servidor e os utilizadores da conversa agora criada.

```
1  (...)
2      use Illuminate\Http\Request;
3      use Illuminate\Support\Facades\Auth;
4      use Illuminate\Support\Facades\Hash;
5      use Illuminate\Support\Facades\Response;
6      use Crypt;
7      use App\ChatRoom;
8      use App\Http\Requests;
9
10     class ChatRoomController extends Controller {
11
12         public function generateChatroom(Request $request) {
13             // get data from the request
14             $data = $request->all();
15
16             // create a new entry
17             $newChatroom = ChatRoom::create([
18                 'chatSSKi' => Crypt::encrypt(str_random(40)),
19                 'chatV'  => $data['timeToExpire']
20             ]);
21
22         // send a response to the user
23         return Response::json([ 'chatroom' => $newChatroom ]);
24     }
25 (...)
```

Listagem 4.3: Criação de um novo *ID_CHAT*

Com o *ID_CHAT* criado, o utilizador pode neste momento dar inicio a uma conversação. Neste contexto podem ser geradas uma ou mais mensagens para cada utilizador em cada *ID_CHAT* criado. As mensagens são cifradas pela aplicação cliente e enviadas para o servidor. O servidor apenas atribui um novo *ID* à nova mensagem e armazena-a na base de dados. A aplicação é incapaz de aceder ao conteúdo das mensagens trocadas entre os utilizadores (*zero knowledge*) porque estas são cifradas com uma chave ($chatSK_i$) que só é do conhecimento das aplicações cliente.

```
(...)
    // iterate all arrays
    foreach ($request->get('chat_ids') as $chatId) {
        $query = ChatRoom::find($chatId)->messages();

        if ($request->has('last_ts')) {
          $query->where('created_at', '>=', $request->get('last_Ts'));
        }

        // append the result to the result array
        $result[$chatId] = $query->get();
     }
      return Response::json([ 'updates' => $result ]);
    }
(...)
```

LISTAGEM 4.4: Obter mensagens associadas ao *ID_CHAT*

Qualquer utilizador pode obter as suas mensagens por intermédio da *route api/getChatRoomUpdates* do controlador *ChatRoomController* (Listagem 4.4). A função *getChatRoomMessages* devolve as mensagens associadas a cada *ID_CHAT*.

Cliente *EkoChat - Ionic*

A aplicação cliente foi desenvolvida com recurso à *framework Ionic* já que esta permite o desenvolvimento de aplicações móveis multi-plataforma, reduzindo assim o tempo necessário para o desenvolvimento das várias versões (*Android*, *IOS*). Afim de validar a solução proposta, implementou-se apenas um protótipo da aplicação cliente que apenas executa em sistemas *Android*. A Figura 4.9 apresenta a vista inicial da aplicação cliente (*front-end*) que solicita a autenticação do utilizador. Esta autenticação (ver Listagem 4.5) é efetuada com um pedido *post* HTTP à *route api/signin* do servidor.

```
(...)
    $http.post('http://172.20.129.78:8000/api/signin', credentials).success(function(response){
        // save the user data
        user = response;

    deferred.resolve(user);
    }).error(function(){
        deferred.reject('Wrong credentials.');
    })
(...)
```

LISTAGEM 4.5: Autenticação do utilizador na aplicação cliente

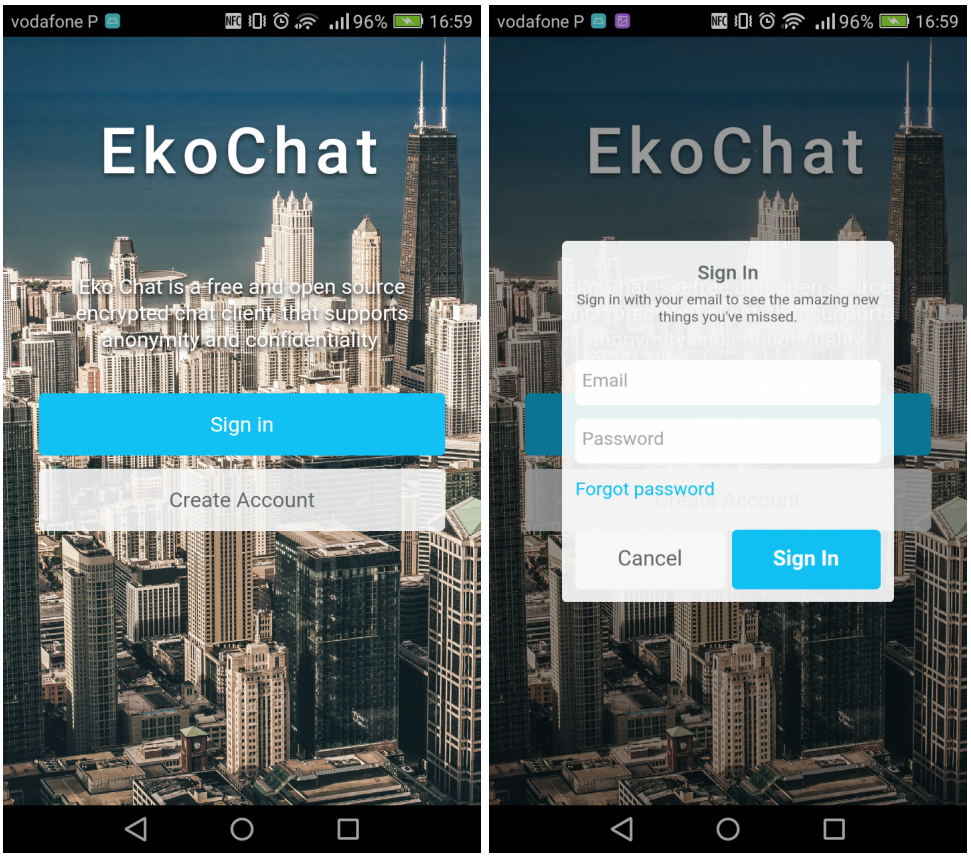

FIGURA 4.9: Vista inicial da aplicação cliente *EkoChat*

Após a autenticação do utilizador, este poderá criar uma nova conversa (um novo *ID_CHAT*), como se pode ver na Figura 4.10. Para tal é efetuado um pedido tipo *post* HTTP para o servidor, fornecendo-lhe a seguinte informação: $userID$, $userT_s$, $chatV$. O servidor, por sua vez, devolve um *chatID* e uma chave simétrica partilhada entre o servidor e os utilizadores desta nova conversa ($chatSSK$).

Após a obtenção da resposta do servidor, continuando a analisar o conteúdo da Listagem 4.6 e para completar a estrutura de dados *ID_CHAT* (ver Tabela 4.1), é gerada uma nova chave simétrica ($chatSK_i$). Esta chave é criada pela aplicação no *smartphone* do utilizador e será utilizada para cifrar todas as mensagens enviadas, no contexto desta conversa, para o servidor.

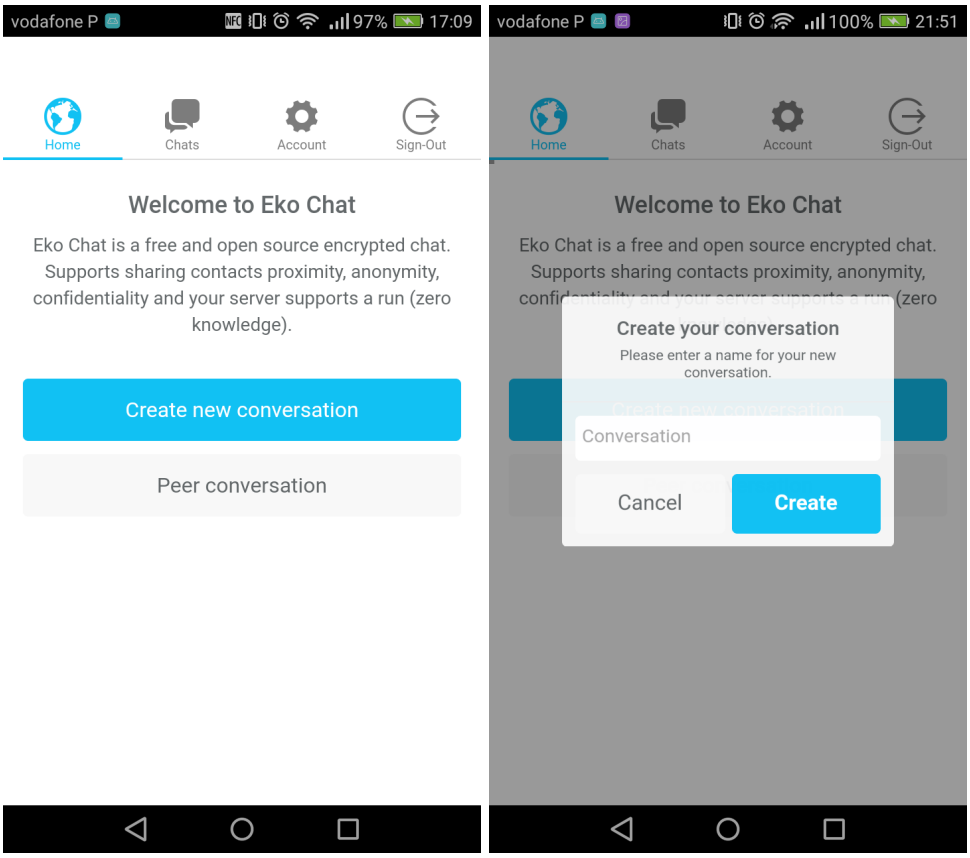

FIGURA 4.10: Vista para criar nova conversação *ID_CHAT* na aplicação cliente *EkoChat*

```
1  (...)
2      $http.post('http://172.20.131.96:8000/api/generateChatroom', data)
3      .success(function(response){
4          // get the new chatroom information
5          var chatroom = response.chatroom
6
7          // generate a new AES key to chypher the shared messages
8          chatroom.chatSKi = generateAESKey()
9
10         // append the name
11         chatroom.name = data.name
12
13         // add a new chat room
14         Chats.add(chatroom)
15     })
16 (...)
```

LISTAGEM 4.6: Criação de um novo *ID_CHAT* na aplicação cliente

O *ngStorage* foi utilizado para guardar localmente a estrutura de dados relativa a cada *ID_CHAT*. A integração de *ngStorage* numa qualquer aplicação *Ionic* é bastante acessível. Sendo que, o armazenamento local é seguro e podem ser armazenadas localmente bastantes quantidades de dados, sem afetar o desempenho da aplicação.

Sempre que um utilizador termina a sua sessão da aplicação, as estrutura de dados *ID_CHAT* são armazenadas localmente.

A Listagem 4.7 apresenta a função de partilha, entre utilizadores, da estrutura de dados *ID_CHAT* via NFC. Em particular, na linha 4, pode ver-se a construção do *payload* a partilhar, que não é mais do que uma estrutura de dados em formato JSON.

```
1  (...)
2      shareChat: function (chat) {
3          // build the payload message to be sent to the other divice
4          var payload = JSON.stringify({ id: chat.id, name: chat.name, chatV: chat.chatV, chatSKi:
       chat.chatSKi, chatSKKi: chat.chatSKKi })
5
6          // create a NFC record
7          var record = ndef.mimeMediaRecord('text/json', nfc.stringToBytes(payload))
8
9          // success hanlder
10         var success = function () {
11             // stop sharing
12             nfc.unshare()
13         }
14
15         // error handler
16         var error = function (reason) {
17         }
18
19         // send the NFC record
20         nfc.share([record], success, error)
21 (...)
```

Listagem 4.7: Partilha da estrutura de dados *ID_CHAT* via NFC entre utilizadores

A Listagem 4.8 mostra a função de envio de uma mensagem, via *post* HTTP, para o servidor. Todas as mensagens enviadas são cifradas. Na linha 8 e 17 é visível o uso da biblioteca *CryptoJS* com AES e recurso à chave $chatSK_i$ e chave $chatSKK_i$ para cifrar as mensagens.

```
1  (...)
2      // create sender information
3      var text = {
4          message: message,
5          userName: user.name
6      }
7      // encrypted text
8      var ciphertext = CryptoJS.AES.encrypt(JSON.stringify(text), chatSKi).toString()
9
10     // create store the sender information
11     var packet = {
12         cipherText: ciphertext,
13         userId: user.id,
14         userTs: new Date().getTime()
15       }
16     // encrypt the message packet
17     var cipherPacket = CryptoJS.AES.encrypt(JSON.stringify(packet), chatSKKi).toString()
18     // make a request to the API
19     $http.post('http://172.20.131.96:8000/api/message', {
20         messageText: cipherPacket,
21         userTs: new Date().getTime(),
22         chatroom_id: id
23
24     }).success(function(response){
25         deferred.resolve()
26     }).error(function(){
27         deferred.reject('An error was occured!')
28     })
29 (...)
```

Listagem 4.8: Envio de uma nova mensagem cifrada

Por último, na Listagem 4.9 pode ver-se a função utilizada para obter as últimas mensagens enviadas para uma determinada conversa. Consiste num pedido tipo *post* HTTP, com periodicidade de 10 segundos, à *route api/getChatRoomUpdates* do servidor (linha 7).

```
(...)
    var chats = {
        userTs: new Date().getTime(),
        chat_ids: self.getAllChatIds()
    }
    setInterval(function () {
    $http.post('http://172.20.131.96:8000/api/getChatRoomUpdates', {
        chatIDs: chats,
        userTs: new Date().getTime(),
        userId: user.id
    })
    .success(function (response) {
        console.log('UPDATES >>>', response)
        // the response constains the follow structure
        // { updates: { chat_id: [ messages ] } }
        processUpdate.call(self, response.updates)
    })
    .error(function () {
        })
     }, 1000 * 10);
    }
(...)
```

Listagem 4.9: Função para atualização das mensagens

4.5 Conclusão

A solução proposta consiste numa aplicação de conversação que visa principalmente a comunicação confidencial, anónima (perante o servidor) e com garantia da identidade dos intervenientes em cada conversação. A solução proposta assenta no modelo cliente/servidor. O servidor foi desenvolvido com recurso à *framework Laravel* e contém os pontos de acesso necessários para a operação dos clientes. A aplicação cliente foi desenvolvida com recurso à *framework Ionic* que permite o desenvolvimento multi-plataforma.

Capítulo 5

Avaliação da Solução

A avaliação da solução proposta consistiu em duas grandes tarefas. Uma primeira tarefa retratou a avaliação funcional e da satisfação dos requisitos identificados inicialmente. O protótipo desenvolvido adota a arquitetura cliente/servidor. O cliente pode ser instalado num *smartphone Android* ou *IOS*. A versão para *IOS* ainda apresenta algumas limitações de funcionamento devido à restrição do uso de NFC a aplicações *Apple Pay*. O servidor é um servidor *web* implementado com a *framework* Laravel.

A segunda tarefa, da análise de segurança da proposta de solução, consiste no tecer de considerações sobre a resistência da solução a ataques ao anonimato e de segurança.

5.1 Avaliação funcional

A avaliação funcional consistiu no teste das várias funcionalidades da proposta de solução, validando a satisfação dos requisitos identificados previamente. A saber, os requisitos que foram identificados podem ser resumidos a: identidade, confidencialidade, anonimato, usabilidade e portabilidade. O primeiro teste da aplicação implicou o estabelecimento da ligação entre o cliente e o servidor. A ligação é estabelecida com o protocolo HTTP. A Figura 5.1 apresenta um excerto do *log* do cliente onde esta é retratada a ligação entre a aplicação cliente e o servidor.

```
connection server API http://192.168.1.2:8000/chat.api/signin                app.js:82
```

Figura 5.1: Registo de ligação do cliente ao servidor

Após estabelecida a ligação, procedeu-se ao registo de utilizadores na solução desenvolvida. A Figura 5.2 representa a vista de registo de um novo utilizador. Esta vista efetua verificações em todos os campos. Como resultado esperado da inserção de utilizadores, devem surgir novas entradas na tabela de utilizadores da base de dados do servidor. A Figura 5.3 apresenta uma lista dos utilizadores inseridos na base de dados em resultado deste teste.

O passo seguinte consistiu no teste ao procedimento de autenticação de utilizadores. Como resposta a um pedido de autenticação de um utilizador com credenciais de acesso válidas, o

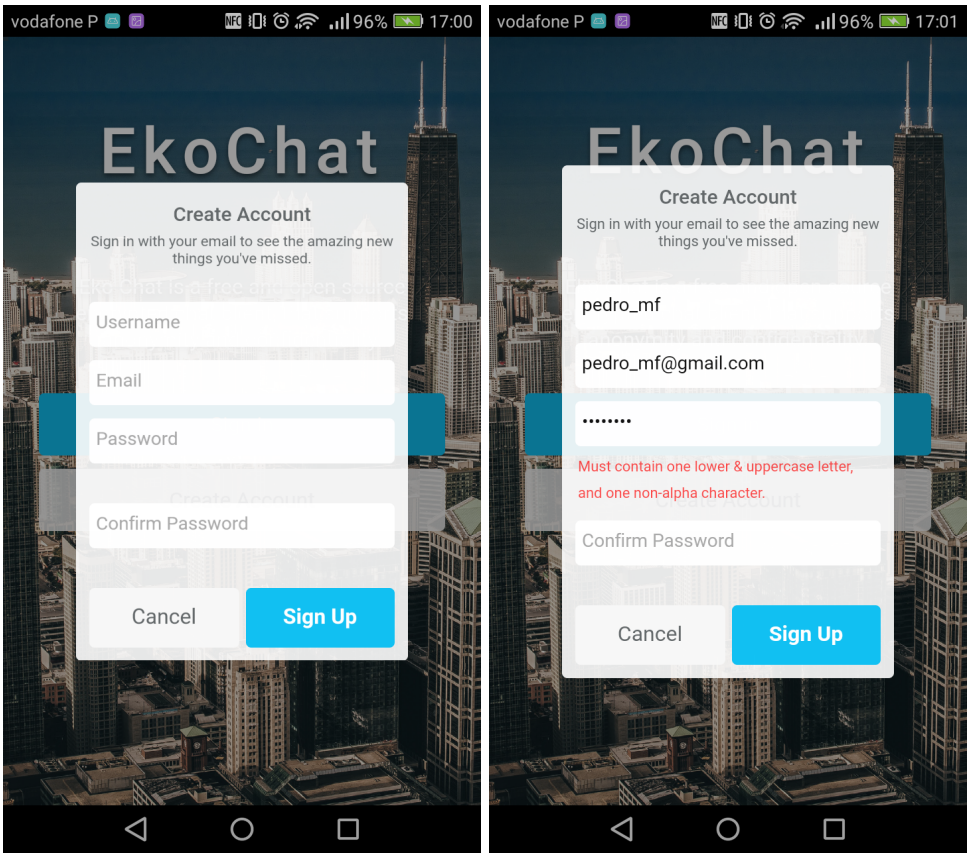

FIGURA 5.2: Vista da aplicação cliente para criação de novo utilizador

name	email	password	remember_token	created_at	updated_at
8080001	8080001@estgf.ipp.pt	$2y$10$e6i1QU0WixQsbqayxZ2d/eJHuH...	NULL	2016-11-17 14:52:18	2016-11-17 14:52:18
8080002	8080002@estgf.ipp.pt	$2y$10$bu4nH.FfGOb/X3ItWi9DneclgiuU...	NULL	2016-11-17 14:52:30	2016-11-17 14:52:30
8080003	8080003@estgf.pp.pt	$2y$10$J9iz1fPoJ5YliY9jJJFIf.PTpX0NgttlJ...	NULL	2016-11-18 16:15:47	2016-11-18 16:15:47
pedro_mf	pedro_mf@gmail.com	$2y$10$YYmbdKs0.ooMh5xJw5YJHu5UY...	NULL	2016-11-19 16:57:21	2016-11-19 16:57:21
8080084	8080084@estgf.ipp.pt	$2y$10$hrvc9Go1TYL0A7XTP86bPOiqfF...	NULL	2016-11-21 11:06:08	2016-11-21 11:06:08
miguel_fo	miguel_fo@estgf.ipp.pt	$2y$10$II00Ngb3Dzez.zslTQA39umQ2G...	NULL	2016-11-21 11:07:49	2016-11-21 11:07:49
fernandes	fernandes.info@estgf.ipp.pt	$2y$10$1yJYF8GzBNze2x4M109ofeXscN...	NULL	2016-11-21 11:08:17	2016-11-21 11:08:17
joaquim	joaquimGM@gmail.com	$2y$10$ecBQXrctz.gBxjT0ZTKETese1lO...	NULL	2016-11-21 11:08:42	2016-11-21 11:08:42
antonioRF	antonio_rf@gmail.com	$2y$10$HpQqvXsj23BA9waFc85QO.fZO...	NULL	2016-11-21 11:08:59	2016-11-21 11:08:59
8080092	8080092@estgf.ipp.pt	$2y$10$vCCfORAirlcc3VBmTlgUAekM/1...	NULL	2016-11-21 11:09:21	2016-11-21 11:09:21

FIGURA 5.3: Tabela de utilizadores da base de dados do servidor

servidor deve gerar um novo *token* de sessão. Como este *token* deve ser único, recorreu-se à *framework* JWT para sua geração. A Figura 5.4 apresenta um exemplo de um *token* devolvido pelo servidor a um utilizador após a sua autenticação com sucesso.

```
Token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vMTkyLj
E2OC4xLjI6ODAwMC9hcGkvc2lnbmluIiwiaWF0IjoxNDc5NzI2NTc5LCJleHAiO
jE0Nzk3MzAxNzksIm5iZiI6MTQ3OTcyNjU3OSwianRpIjoiYmM1Y2I3ZTY4NGJl
OTdkMjg5MzYxMDAwNGMyMTczOTIiLCJzdWIiOjd9.fIoAwQSa2T4qu2-
9dno5U7JoJs5fyZrqUgn5fbWx6qg
```

FIGURA 5.4: *Token* de utilizador gerado com JWT

A Figura 5.5 apresenta um registo da criação de uma nova conversa que implica a geração de uma chave criptográfica ($chatSKK_i$), guarda-la numa estrutura de dados *ID_CHAT* e a sua devolução por parte do servidor. A chave criptográfica ($chatSKK_i$) é uma chave simétrica

partilhada entre utilizadores e o servidor. A chave criptográfica $chatSK_i$ é uma segunda chave simétrica, gerada no *smartphone* do utilizador, a juntar ao *chatID* e restante informação no *ID_CHAT*. Este é um processo que se revelou relativamente rápido.

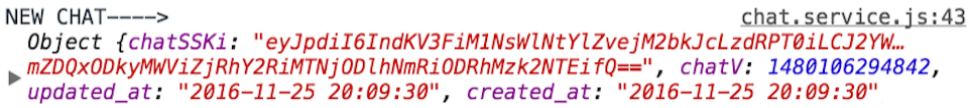

FIGURA 5.5: Criação de um novo *ID_CHAT*

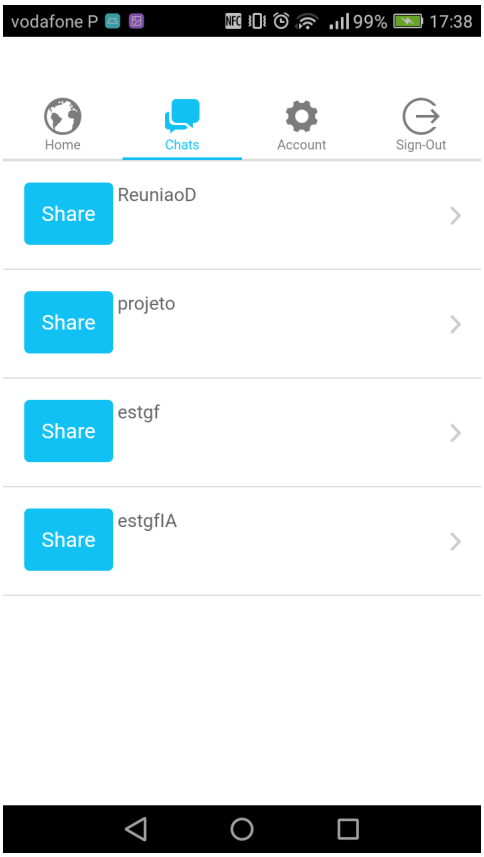

FIGURA 5.6: Vista da aplicação cliente para criação de uma nova conversação

A Figura 5.7 apresenta um excerto de conversações inseridas na base de dados da aplicação. Após o teste de criação de novas conversações, foram efetuadas várias trocas via NFC das respetivas estruturas de dados de descrição de conversas *ID_CHAT*. Esta interação serve para verificar a identidade dos contactos e para a troca do material criptográfico necessário para decifrar as mensagens. Desta maneira, a aplicação garante, aos intervenientes em cada conversação, a identidade dos mesmos.

O próximo teste foca-se na verificação da confidencialidade das mensagens trocadas. A aplicação desenvolvida não é capaz, em circunstância alguma, de aceder ao conteúdo das mensagens trocadas entre os utilizadores. As mensagens são cifradas E2E. A Figura 5.9, apresenta uma parte das mensagens armazenadas na base de dados da aplicação servidor *EkoChat*.

A Figura 5.10 apresenta duas vistas da aplicação cliente. Do lado esquerdo surge a vista que lista as conversações existentes no *smartphone*. A cada elemento desta lista, corresponde uma

chatV	created_at	updated_at	chatSSKi
1479580149172	2016-11-19 18:00:38	2016-11-19 18:00:38	eyJpdiI6ImR6TWNjeTVXaDBQSFBmc21uYnBEK1E9PSIsInZhbHVlIjoialVXT...
1479580414618	2016-11-19 18:22:41	2016-11-19 18:22:41	eyJpdiI6Ingrcm1EVFNkRzRTSmdFZXNKUmVrVWc9PSIsInZhbHVlIjoiekJZa...
1479727787261	2016-11-21 11:10:06	2016-11-21 11:10:06	eyJpdiI6Im9KcmVsUkJlNlRCQUhXTmhkTExUTWc9PSIsInZhbHVlIjoiYUM0...
1479727803411	2016-11-21 11:10:16	2016-11-21 11:10:16	eyJpdiI6InJTTjRGMGt1RWFZbVlwa2RoZHZ2Q1E9PSIsInZhbHVlIjoicDNTU...
1479727819074	2016-11-21 11:10:25	2016-11-21 11:10:25	eyJpdiI6IlwvbkxZcVY4TGR0SU50SjZjdkRtdEpRPT0iLCJ2YWx1ZSI6IlRuUjR...
1479727839034	2016-11-21 11:10:35	2016-11-21 11:10:35	eyJpdiI6IkYyUGtCb1BvVlwvditaS1BaTTJxczBBPT0iLCJ2YWx1ZSI6IjIwV0R...
1479727865953	2016-11-21 11:10:50	2016-11-21 11:10:50	eyJpdiI6InVzcUErWnVNMDhjSjVtVFE4cXFSNGc9PSIsInZhbHVlIjoiZkFBYUF...
1479727869808	2016-11-21 11:10:53	2016-11-21 11:10:53	eyJpdiI6InhhblIlMXJYelRhQVdvN0RnUGZHXC93PT0iLCJ2YWx1ZSI6ImZV...
1479727873289	2016-11-21 11:10:55	2016-11-21 11:10:55	eyJpdiI6IkkxbGZ2VHdCa1pmblNJdWhwTGVYSFE9PSIsInZhbHVlIjoiUWcrc...

FIGURA 5.7: Lista de conversas da base de dados do servidor

```
ciphertext >>>                                          chat.factory.js:133
U2FsdGVkX1+lbrpGu9L6ww0zlGFlz01/b8Ds6W+6edY=
```

FIGURA 5.8: Exemplo de mensagem cifrada

messageText	created_at	updated_at
U2FsdGVkX19wqJQ4ZEhl/cfFcyPjvsQTPrFX2piBW7cMaAH+Z3AVxokgk3v9...	2016-11-19 18:00:52	2016-11-19 18:00:52
U2FsdGVkX19lLI6jW8smoG1by2s01iOKawp3pAsuccen6eUJ2vV8DW6YLUo...	2016-11-19 18:00:57	2016-11-19 18:00:57
U2FsdGVkX1/FFbJFLWkjPQRr/u1O4OaUZcGIMvcrpdmw/OoyuTiuR0Dzf5F...	2016-11-19 18:01:01	2016-11-19 18:01:01
U2FsdGVkX19um296SAud2LDheUhZlbbZiOqSOynd1y+QcIxXSpLvEhcHWa...	2016-11-19 18:01:01	2016-11-19 18:01:01
U2FsdGVkX1+eoWCnarZhIccyhA7ATnrOnKy+eV1chycXaPimJEh80EnAHk5...	2016-11-19 18:01:06	2016-11-19 18:01:06
U2FsdGVkX19lG+hgrzc03/8mlfpHWkXwNevW3xQ/px9CJ+og7RyLSVu0M...	2016-11-19 18:01:14	2016-11-19 18:01:14
U2FsdGVkX1/UQ0mRnid90RPN0GHfpZot5GOmTCXrlSjZkcu5JTi/WQx8yn...	2016-11-19 18:01:19	2016-11-19 18:01:19
U2FsdGVkX187VUpDosbd1QlETaN4Lo22d9xhIaACz8v6B30LpggAbDhYP...	2016-11-19 18:01:31	2016-11-19 18:01:31
U2FsdGVkX182KeMeuGwUrH2FPan7tAflWuIyR1dzwCcna4VOklN6lDQapp...	2016-11-19 18:23:28	2016-11-19 18:23:28
U2FsdGVkX1+hOEbcs+RsZDtXzgNiTyErA6QBHLKsCxPbSeGLQkku8kQzQ1...	2016-11-19 18:23:34	2016-11-19 18:23:34
U2FsdGVkX187Qe1uZEPL2OlwhyvFfRtKa+ZXWMfisCvlhBoPB95YBYw2SKv...	2016-11-21 11:11:10	2016-11-21 11:11:10
U2FsdGVkX19flyb9lPXDLRkg9gmflRJbpvbADbX5ClZ6ZkWrASwLHzVO+6iK...	2016-11-21 11:11:14	2016-11-21 11:11:14
U2FsdGVkX1/OTn7Xoo5f+3UGy4umlCUlJd5XfqSDxfmVyXgXgiD+aIOtFD...	2016-11-21 11:11:17	2016-11-21 11:11:17
U2FsdGVkX19YSoV7lzBGmcd3HzVwNeRSu5LlCJXqaxo8F0o7VOUk7NFLNi...	2016-11-21 11:11:27	2016-11-21 11:11:27
U2FsdGVkX1+UJi8ZWC1vJjmXJ9zou9+sD//mCifdHTQ8qeS1HoEcd4eYr4o...	2016-11-21 11:11:37	2016-11-21 11:11:37
U2FsdGVkX19baUwqTS24jQNmvJ66jkeDFhGQt7YJydB0HbcIMsyv5eSq82s...	2016-11-21 11:11:47	2016-11-21 11:11:47
U2FsdGVkX1/LTsbKqOqttuvlW/eru5+RY/TyZAjwK+U9TfIt/j0JSoCR/5AvL...	2016-11-21 11:11:50	2016-11-21 11:11:50
U2FsdGVkX184XIOmmqH/kxHQNKLBpt1KcDWiPjWMNrX7cZx3yOJXZJgyP6...	2016-11-21 11:11:59	2016-11-21 11:11:59
U2FsdGVkX19Z6OGuN5iWj9RHZjsU0zo3ZWJSy41ykz9honnraVAJp5GTqMf...	2016-11-21 11:12:02	2016-11-21 11:12:02
U2FsdGVkX1+jeEiGQ+iFRvJ8G/VMnygyMT7SvpAYyZ26fUksc6iUZ/EVHN0...	2016-11-21 11:12:03	2016-11-21 11:12:03
U2FsdGVkX1+mMM6oz92WekJ3ZD4VPThASpM1h0YB3ERiRvtGjAwJ3Y4NS...	2016-11-21 11:12:06	2016-11-21 11:12:06
U2FsdGVkX1/J1jZwofPAPbQVuvMF0NsJGpOpiTuEQAHkhY88ohi4cTSkLuI6...	2016-11-21 11:12:19	2016-11-21 11:12:19

FIGURA 5.9: Lista de mensagens da base de dados do servidor da aplicação

estrutura de dados *ID_CHAT*. Do lado direito surge a vista de conversa onde são apresentadas todas as mensagens de uma conversão.

Por último, e para demonstrar a operação da aplicação cliente em contexto multiplataforma, são apresentados na Figura 5.11 as vistas de inicio de execução da aplicação em *Andorid* e *IOS*. Como a aplicação foi desenvolvida em *Ionic*, e esta *framework* permite criar soluções independentes da plataforma, a aplicação pode ser configurada para executar e trocar informações com diferentes *smartphones*.

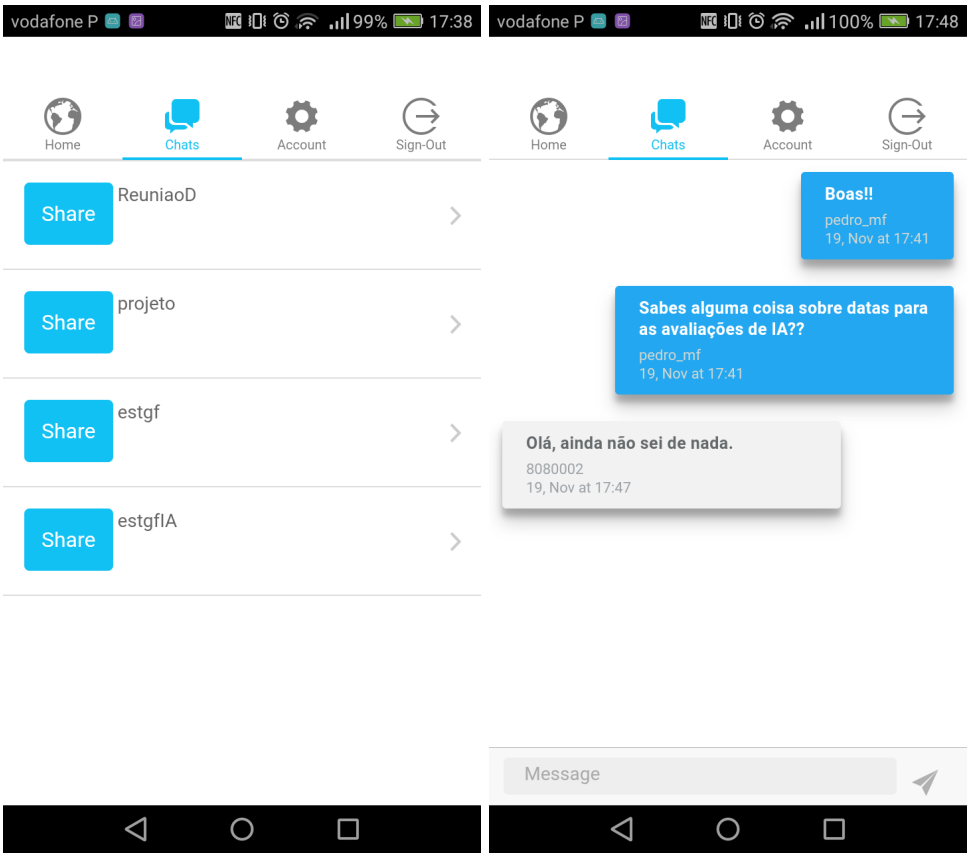

FIGURA 5.10: Vista da aplicação cliente da lista de conversações e de mensagens trocadas

5.2 Análise de segurança

A confidencialidade das mensagens trocadas entre cliente e servidor é obtida por meio da cifra E2E. A chave utilizada para cifrar mensagens em conversas, a chave SK_x, é gerada num dos extremos e guardada localmente, sendo depois trocada entre utilizadores com tecnologias de comunicação de curto alcance (*NFC*). A chave SK_x não é enviada para o servidor. As restantes comunicações entre o utilizador e o servidor são protegidas de várias formas. Para cada conversação existe uma segunda chave, a chave SSK_x, que é do conhecimento dos participantes numa conversa e do servidor e serve para também auxiliar na proteção da confidencialidade das mensagens trocadas. Por último, e sendo um serviço assente em tecnologia *web*, o servidor deve ser disponibilizado apenas em modo seguro (*HTTPS*).

A solução proposta garante anonimato dos utilizadores através da estrutura de dados *ID_CHAT*. A identidade do utilizador no envio de mensagens é totalmente desconhecida ao servidor, cada mensagem é cifrada e apenas os utilizadores tem acesso ao material criptográfico que permite ler ou escrever mensagens. Os identificadores de utilizador que o servidor conhece são utilizados para evitar a identificação unívoca de utilizadores.

A personificação pode ser explorada por indivíduos menos escrupulosos, ou até mesmo criminosos, para levar terceiros a acreditar que uma pessoa é outra e obter daí algum benefício. Este

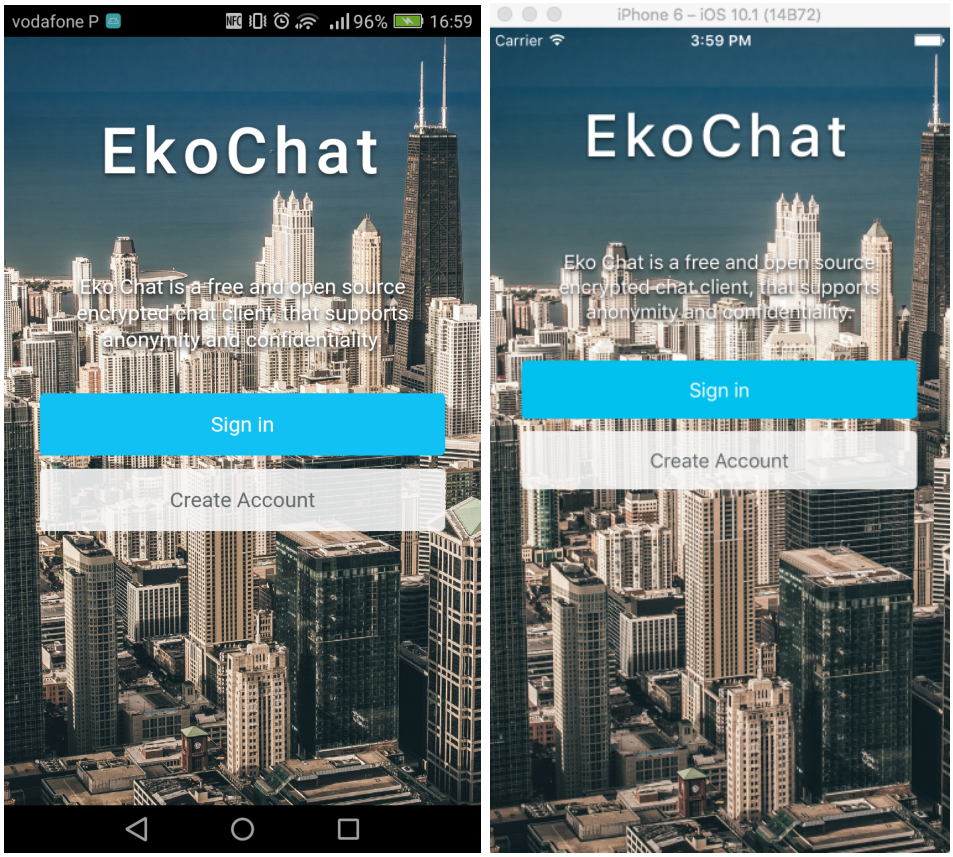

FIGURA 5.11: Execução em *Andorid* e *IOS*

problema é agravado com existem utilizadores menores de idade, ou utilizadores menos habilitados com as novas tecnologias de informação. A solução proposta resolve este problema por requerer uma interação prévia entre os utilizadores para que se dê a conversação. Esta interação consiste na troca da estrutura de dados *chatID* por uma tecnologia de comunicação de curto alcance.

O servidor, ainda na persecução do objetivo do anonimato, opera em modo *zero knowledge*. Este modo é conseguido associando um identificador a cada conversação entre utilizadores e evitando que o servidor aceda à chave de cifra das mensagens. Assim, o servidor apenas facilita a partilha de mensagens cifradas entre os utilizadores de uma conversação. O servidor é incapaz de decifrar as mensagens trocadas.

5.3 Conclusão

A solução proposta satisfaz todos os requisitos previamente identificados. Garante tanto o anonimato dos utilizadores como consegue garantir, do lado do servidor, que um determinado utilizador pode enviar mensagens para uma conversação. O servidor não é capaz de aceder ao conteúdo das mensagens trocadas entre os utilizadores. Todas as mensagens trocadas são cifradas E2E, garantindo a sua confidencialidade. O material criptográfico necessário para ler ou escrever mensagens trocadas com os restantes utilizadores é armazenado no *smartphone*. A

solução proposta pelo facto de ser desenvolvida com a *framework Ionic*, permite a sua execução de forma consistente e independente de plataforma.

Capítulo 6

Conclusão

Desenvolver uma aplicação de troca de mensagens segura é uma tarefa árdua. Mensagens seguras não dependem só de uma cifra forte. A superfície de ataque da solução deve ser limitada ao máximo, de modo a limitar eventuais impactos no anonimato e na confidencialidade das mensagens trocadas. A solução proposta, de nome *EkoChat*, visa a comunicação anónima, confidencial e resistente à utilização de falsas identidades. A ideia assenta numa aplicação que permite a troca segura de mensagens entre utilizadores verificados previamente.

6.1 Contribuição

A principal contribuição do trabalho desenvolvido consiste numa aplicação de *chat* segura para *smartphones* com suporte para anonimato e confidencialidade e resistente à utilização de identidades falsas. Tal é conseguido com recurso à partilha prévia de material criptográfico por NFC que nunca chega a ser do conhecimento do servidor. Das várias aplicações de conversação para *smartphones*, nenhuma outra solução obriga a uma interação prévia de garantia de identidade para que se dê a comunicação. Facto este que motiva a consideração da solução proposta como uma contribuição inovadora.

6.2 Trabalho futuro

No decorrer do trabalho apresentado nesta tese, foram identificados alguns avanços que podem ser assumidos como direções futuras de pesquisa e de aplicação. Um dos avanços passa por implementar o suporte para partilha de contactos *ID_CHAT* por *Bluetooth* em sistemas baseados em *IOS*. Outro consiste em tornar esta troca mais robusta, gerando também um código no *smartphone* para confirmação do mesmo por parte do recetor, aquando da troca do material criptográfico. Finalmente, evoluir a solução para permitir de chamadas de voz com o mesmo nível de segurança.

Bibliografia

[1] Anders Andersen and Randi Karlsen. Experimenting with Instant Services Using NFC Technology, 2012.

[2] AngularJS. What Is Angular? `https://docs.angularjs.org/guide/introduction`.

[3] O'Reilly & Associates. JavaScript: The Definitive Guide (6th ed.). Last accessed 20 June 2016.

[4] Melani Au, Jeanne Lam, and Radar Chan. Social media education: Barriers and critical issues. In *Technology in Education. Transforming Educational Practices with Technology*, pages 199–205. Springer, 2015.

[5] Auth0. Introduction to JSON Web Tokens. `https://jwt.io/introduction/`. Last accessed 20 September 2016.

[6] Calderbank, Michael. The RSA Cryptosystem: History, Algorithm, Primes, 2007.

[7] CodeProject. JSON vs. XML: verbosity. `http://www.codeproject.com/Articles/604720/JSON-vs-XML-Some-hard-numbers-about-verbosity`. Last accessed 4 July 2016.

[8] Vedat Coskun, Kerem Ok, and Busra Ozdenizci. *Professional NFC application development for android*. John Wiley & Sons, 2013.

[9] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

[10] Cropf, Robert A. *Ethical Issues and Citizen Rights in the Era of Digital Government Surveillance*. IGI Global, Feb 2, 2016.

[11] Demircioglu, Murat and Taskın, Halil Kemal and Sarımurat, Salim. Security analysis of the encrypted mobile communication applications, 2014.

[12] Electronic Frontier Foundation. Secure Messaging Scorecard. Which apps and tools actually keep your messages safe? Last accessed 11 September 2016.

[13] Electronic Frontier Foundation, Julia Angwin, Joseph Bonneau. Secure Messaging Scorecard. `https://www.eff.org/secure-messaging-scorecard`, 2014.

[14] F. Breitinger F. Karpisek, I. Baggili. Whatsapp network forensics: Decrypting and understanding the whatsapp call signaling messages. *Digital Investigation*, 15:110 – 118, 2015. Special Issue: Big Data and Intelligent Data Analysis.

[15] Maximiliano Firtman. HTML5. http://mobilehtml5.org. Last accessed 11 June 2016.

[16] NFC Forum. Near Field Communication Forum. http://nfc-forum.org/, 2004. Last accessed 14 March 2016.

[17] NFC Forum. What It Does, 2004.

[18] Gaurav Rathee. How WhatsApp Works. http://digitalperiod.com/explore-whatsapp-clock-sign-and-tick/, 2015.

[19] Andy Greenberg. Hacker Lexicon: What Is End-to-End Encryption?, WIRED, 2015. Last accessed 17 March 2016.

[20] Ian Hickson. HTML5 Introduction. (1999-2012). http://www.w3schools.com/html5/html5_intro.asp. Last accessed 11 June 2016.

[21] Wickr Inc. How Wickr's Encryption Works. https://www.wickr.com/security/how-it-works. Last accessed 17 February 2016.

[22] Wickr Inc. Wickr. https://www.wickr.com/.

[23] Ecma International. Introducing JSON. http://www.json.org/. Last accessed 3 July 2016.

[24] Ionic. Ionic Documentation Overview. http://ionicframework.com/docs/overview/.

[25] JScripters.com. JavaScript: Advantages and Disadvantages. http://www.jscripters.com/javascript-advantages-and-disadvantages/. Last accessed 22 June 2016.

[26] Robert Davidson & Akiba Kevin Townsend. Getting Started with Bluetooth Low Energy. https://www.safaribooksonline.com/library/view/getting-started-with/9781491900550/ch01.html, 2016. Last accessed 14 August 2016.

[27] Max Lynch. New Supported Android versions. http://blog.ionic.io/market-share-movement-android/. Last accessed 20 September 2016.

[28] Shawn McCool. *Laravel Starter*. Packt Publishing Ltd, 2012.

[29] Daria Nurtdinova et al. Security in mobile messaging, 2016.

[30] Open Whisper Systems. Signal protocol. https://whispersystems.org/blog/license-update/. Last accessed 11 September 2016.

[31] Oracle. Analysis of JSON use cases compared to XML. https://blogs.oracle.com/xmlorb/entry/analysis_of_json_use_cases. Last accessed 1 July 2016.

[32] Taylor Otwell. Laravel. https://laravel.com/docs/4.2/introduction. Last accessed 6 September 2016.

[33] Simon Pieters. HTML5 Differences from HTML4, 2010. Last accessed 7 August 2016.

[34] Rescorla, Eric. Diffie-hellman key agreement method, 1999.

[35] Ben Ripkens. Ionic: An AngularJS based framework on the rise. https://blog.codecentric.de/en/2014/11/ionic-angularjs-framework-on-the-rise/. Last accessed 24 September 2016.

[36] Saint-Andre, Peter. Extensible messaging and presence protocol (XMPP): Core, 2011.

[37] Raphaël Saunier. *Getting started with Laravel 4*. Packt Publishing Ltd, 2014.

[38] Bluetooth SIG. Basic Rate/Enhanced Data Rate (BR/EDR). https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/br-edr. Last accessed 10 September 2016.

[39] Bluetooth SIG. Bluetooth. https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth. Last accessed 10 August 2016.

[40] Bluetooth SIG. Bluetooth low energy (LE). https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy. Last accessed 11 September 2016.

[41] Sinisa. Architecture of Laravel 5.2 Applications. http://www.sinisalekovic.com/blog/architecture-of-laravel-5-2-applications. Last accessed 6 September 2016.

[42] STMicroelectronics. ISO/IEC 14443 A&Band JIS-X 6319-4. Last accessed 12 December 2015.

[43] Leo Sun. Facebook Inc.'s WhatsApp Hits 900 Million Users: What Now? The Motley Fool. http://www.fool.com/investing/general/2015/09/11/facebook-incs-whatsapp-hits-900-million-users-what.aspx, 2015.

[44] Maks Surguy. History of Laravel PHP framework, Eloquence emerging, 2013.

[45] Open Whisper Systems. Advanced cryptographic ratcheting. https://whispersystems.org/blog/advanced-ratcheting/. Last accessed 28 September 2016.

[46] Tactical Technology Collective and Front Line Defenders. TEXTSECURE FOR ANDROID, security in-a-box. https://securityinabox.org/en/guide/textsecure/android, 2009.

[47] D. Talmesio. One SIM per person on the planet, but still too many unconnected. http://www.ovum.com/press_releases/one-sim-per-person-on-the-planet-butstill-too-many-unconnected/. Last accessed 9 February 2016.

[48] TechCrunch. AOL. Talk Private To Me: Free, Worldwide, Encrypted Voice Calls With Signal For iPhone. Last accessed 13 September 2016.

[49] Telegram. Mtproto mobile protocol.

[50] Telegram. Telegram.

[51] The Zfone Project. Exactly how does Zfone and ZRTP protect against a man-in-the-middle (MiTM) attack? Last accessed 13 September 2016.

[52] Threema GmbH. Threema.

[53] Threema GmbH. Threema Cryptography Whitepaper. https://threema.ch/en/faq. Last accessed 10 September 2016.

[54] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jorg Schwenk, Thorsten Holz. How Secure is TextSecure?, 2014.

[55] Devharsh Trivedi and M Tech Student. Near Field Communication, 2015.

[56] Tutorialspoint. AngularJS. http://www.tutorialspoint.com/angularjs/angularjs_overview.htm. Last accessed 6 June 2016.

[57] TutorialsPoint. Ionic Environment Setup. https://www.tutorialspoint.com/ionic/ionic_environment_setup.htm. Last accessed 20 September 2016.

[58] TutorialsPoint. Ionic Overview. https://www.tutorialspoint.com/ionic/ionic_overview.htm. Last accessed 21 September 2016.

[59] WHAnonymous. Funxmpp-protocol. https://github.com/WHAnonymous/Chat-API/wiki/FunXMPP-Protocol, 2015.

[60] Wikimedia Commons. Axolotl_ratchet_scheme. Last accessed 15 September 2016.

[61] Sahat Yalkabov. Login with OAuth 1.0. https://github.com/sahat/satellizer#-login-with-oauth-10.

[62] Sahat Yalkabov. Login with OAuth 2.0. https://github.com/sahat/satellizer#-login-with-oauth-20.

[63] Sahat Yalkabov. Satellizer Token-based AngularJS Authentication. https://github.com/sahat/satellizer.