Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

PROTOCOLO SEGURO DE MENSAGENS INSTANTÂNEAS

JOÃO BERNARDO GOMES JORGE

Leiria, março de 2022

Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

PROTOCOLO SEGURO DE MENSAGENS INSTANTÂNEAS

JOÃO BERNARDO GOMES JORGE
Número: 2190373

Projeto realizado sob orientação da Professora Doutora Marisa Maximiano (marisa.maximiano@ipleiria.pt) e do Professor Ricardo Gomes (ricardo.p.gomes@ipleiria.pt).

Leiria, março de 2022

AGRADECIMENTOS

Começo por agradecer à minha família por todo o suporte dado não só ao longo deste percurso académico mas também ao longo da minha vida, através dos valores e educação que me foram transmitindo. Agradeço, também, aos professores que me foram educando ao longo da minha vida académica, pois sem o conhecimento e educação partilhados por eles e pela minha família não seria a pessoa que sou hoje.

Um agradecimento especial à minha namorada que esteve sempre ao meu lado nos melhores momentos nos de mais pressão, tentando que não perdesse o foco do que realmente é importante.

Pretendo agradecer, ainda, aos meus colegas de trabalho que me deram a oportunidade de conciliar a minha profissão com o Mestrado.

Agradeço não só aos professores da Licenciatura em Engenharia Informática, que me transmitiram o conhecimento que me sustenta no meu dia a dia laboral, mas também aos professores do Mestrado em Cibersegurança e Informática Forense que fomentaram ainda mais o meu gosto e interesse na vasta área da Cibersegurança.

Por fim, estou muito grato pelo apoio dos meus professores orientadores, Marisa Maximiano e Ricardo Gomes, que nunca desistiram deste projeto e me deram a mão para juntos chegarmos ao final desta etapa do meu percurso académico. A sua ajuda foi incansável e chave para o sucesso deste trabalho.

RESUMO

A sociedade usufrui bastante de aplicações de Mensagens Instantâneas (MI) para comunicar diariamente entre si. Desde simples conversas entre duas pessoas ou mesmo num grupo de trabalho. Assim, é essencial garantir que a comunicação efetuada aplica políticas de segurança de forma a mitigar eventuais ataques a essa informação.

Paralelamente, a descentralização também se tem demonstrado uma tendência nos últimos tempos, sendo que uma das formas de aplicar este conceito nas aplicações de MI consiste na utilização da tecnologia Peer-To-Peer (P2P). Esta divide-se em vários tipos, no entanto, a forma mais pura de aplicar esta tipo de arquitetura é através de Pure Peer-To-Peer (PP2P).

Deste modo, foi desenvolvido um «protocolo» baseado no já existente *Signal* que difere na não utilização de uma terceira entidade reguladora da comunicação, o servidor onde são hospedadas as chaves públicas dos utilizadores que fazem uso da aplicação e por onde as mensagens são encaminhadas.

Em conjunto com o «protocolo» foi desenvolvida uma aplicação de *chat* PP2P de forma a que os utilizadores comuniquem diretamente entre si, fazendo uso do «protocolo» desenvolvido no âmbito deste trabalho, cujo objetivo é demonstrar o seu funcionamento, garantindo a implementação de algumas das medidas de segurança também demonstradas pelo protocolo *Signal*. Este foi implementado dando uso à linguagem JavaScript (JS) convergindo com a construção da aplicação com recurso à *framework ElectronJS*.

ABSTRACT

Society makes great use of Instant Messages (IM) applications to communicate with each other on a daily basis. From simple conversations between two people or even in a work group. Thus, it is essential to ensure that the communication carried out applies security policies in order to mitigate any attacks on that information.

At the same time, decentralization has also been a trend, and one of the ways to apply this concept in IM applications is the use of Peer-To-Peer (P2P) technology. This is divided into several types, however, the purest way to apply this type of architecture is through Pure Peer-To-Peer (PP2P).

In this way, a «protocol» was developed based on the existing Signal, which differs in the non-use of a third entity that menages the communication, the server, where the public keys of the users who use the application are hosted and through which the messages are forwarded.

Simultaneously with this «protocol», a PP2P chat application was developed so that users communicate directly with each other using the «protocol» developed within the scope of this work, whose objective is to demonstrate its operation, ensuring the implementation of some of the security measures that are also demonstrated by the Signal protocol. This was implemented using the JavaScript (JS) language merging with the construction of the application using the ElectronJS framework.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AEAD	Authenticated Encryption with Associated Data.
API	Application Programming Interface.
APNS	Apple Push Notification Service.
BIOS	Basic Input/Output System.
bit	Digito binário.
Byte	Unidade de informação digital composta por oito bits.
DHC	Diffie-Hellman Chain.
DHCP	Dynamic Host Configuration Protocol.
DNS	Domain Name System.
DR	Double Ratchet.
ECDSA	Elliptic Curve Digital Signature Algorithm.
EMS	Element Matrix Services.
ENISA	European Union Agency for Cybersecurity.
ESTG	Escola Superiro de Tecnologia e Gestão.
FTP	File Transfer Protocol.
GDPR	General Data Protection Regulation.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
HTTPS	Hypertext Transfer Protocol Secure.
IM	Instant Messages.
IP	Internet Protocol.

IPC	Inter-Process Communication.
IPLEIRIA	Instituto Politecnico de Leiria.
ISP	Internet Service Provider.
JS	JavaScript.
KDF	Key Derivation Function.
MAC	Message Authentication Code.
MI	Mensagens Instantâneas.
MITM	Man In The Middle.
OTP	One Time Password.
P2P	Peer-To-Peer.
PP2P	Pure Peer-To-Peer.
RC	Receiving Chain.
RGPD	Regulamento Geral de Proteção de Dados.
SC	Sending Chain.
SO	Sistema Operativo.
SRC	Segurança de Redes de Computadores.
SSL	Secure Sockets Layer.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
TXT	Ficheiro texto.
UC	Unidade Curricular.
UI	User Interface.
X3DH	Triple Diffie-Hellman.

1

INTRODUÇÃO

1.1 OBJETIVOS

Este documento reflete o projeto realizado no âmbito da Unidade Curricular (UC) de Projeto Informático do Mestrado em Cibersegurança e Informática Forense da Escola Superior de Tecnologia e Gestão (ESTG) do Instituto Politécnico de Leiria (IPLEIRIA), tendo sido orientado pela Professora Doutora Marisa Maximiano e pelo Professor Ricardo Gomes. O principal objetivo do mesmo baseia-se no estudo dos protocolos das aplicações de Mensagens Instantâneas (MI) mais comuns e na criação de uma prova de conceito através da utilização de componentes do protocolo *Signal* [1] de forma a desenvolver uma aplicação de *chat* com arquitetura Pure Peer-to-Peer (PP2P) [2]. Pretende-se verificar a possibilidade da utilização dos principais componentes do protocolo *Signal* sem que exista recurso a uma terceira entidade, reguladora e responsável pela distribuição das chaves entre as entidades, bem como do encaminhamento das mensagens. Para tal, propõe-se implementar a aplicação em contexto multiplataforma, com recurso à *framework ElectronJS*, com a comunicação entre instâncias da mesma usando Web Sockets.

1.2 MOTIVO

Nos dias de hoje, cada vez mais se tem verificado um crescimento significativo de utilizadores nos serviços de comunicação tradicionais, que na maioria dos casos são controlados por terceiros, em que não há garantias da sua segurança e nem privacidade dos dados dos seus utilizadores.

Um dos elos mais fracos dos serviços de comunicação tradicionais é a arquitetura, que necessita de uma terceira entidade, responsável por estabelecer, autenticar e criar ligações. A adoção deste tipo de arquitectura possui as suas vantagens, mas adiciona pontos de falha, que podem ser suscetíveis a vários ataques. Assim, é necessário aplicar medidas/políticas de segurança que permitam mitigar essas falhas.

A definição de cibersegurança, segundo [3], é a aplicação de normas/políticas para proteger programas, redes e sistemas de ataques digitais.

Estes ataques digitais, por norma, distribuem-se em:

- Acesso indevido, manipulação ou destruição de informação sensível, muitas vezes relacionada com dados pessoais;
- Extorção de valor financeiro;
- Disrupção/interrupção de serviço.

Assim, a aplicação de políticas de segurança de informação são fundamentais para evitar/dificultar os ciberataques, que, segundo [3], são mais difíceis de prevenir pois existem mais dispositivos do que pessoas e os atacantes vão inovando os métodos de ataque.

A Figura 1 sintetiza as principais ameaças reportadas pela European Union Agency for Cybersecurity (ENISA) durante o período compreendido entre abril de 2020 e meados de julho de 2021.

Figura 1: ENISA: Principais ameaças de 2021 [4]

É possível verificar que uma das principais ameaças reportadas pela ENISA é contra a disponibilidade e integridade da informação. Estas duas características

de segurança são precisamente duas, das diversas características, que este projeto procura implementar na arquitetura da aplicação PP2P.

Na Figura 2 é possível verificar um aumento em todos os pontos representados que vão desde a população que frequenta o «mundo digital», utilizadores de telemóvel únicos (independentemente do número de telemóveis que possuem, apenas uma utilização conta para a estatística), de utilizadores de Internet e de utilizadores ativos nas redes sociais.

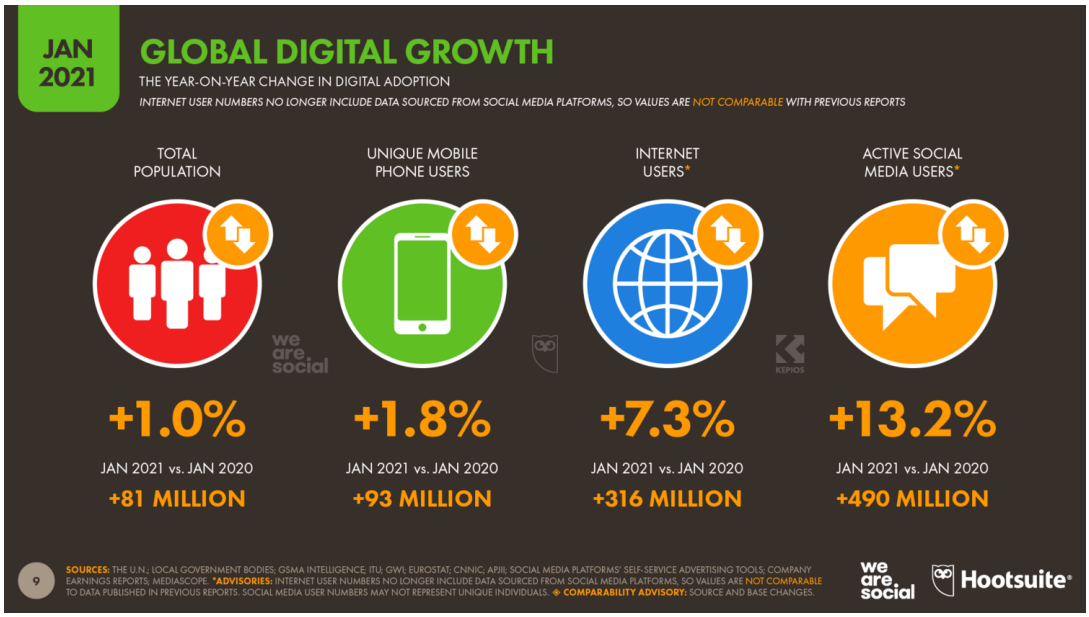

Figura 2: Crescimento do «mundo digital» [5]

Estes registos indicam claramente um crescimento do «mundo digital». Este aumento tem de ser paralelo à melhoria e à maior utilização de medidas de segurança da informação por parte das aplicações *web* ou mesmo de dispositivos móveis de forma a mitigar eventuais ataques informáticos às entidades detentoras dos dados das pessoas.

Atualmente, a utilização das redes sociais é generalizada, isto é, várias pessoas de várias partes do mundo utilizam as redes sociais para partilhar informações ou mesmo comunicar. Segundo o relatório anual de 2021 [5], existem cerca de 4,2 mil milhões de utilizadores ativos nas redes sociais, tal como se pode verificar na Figura 3.

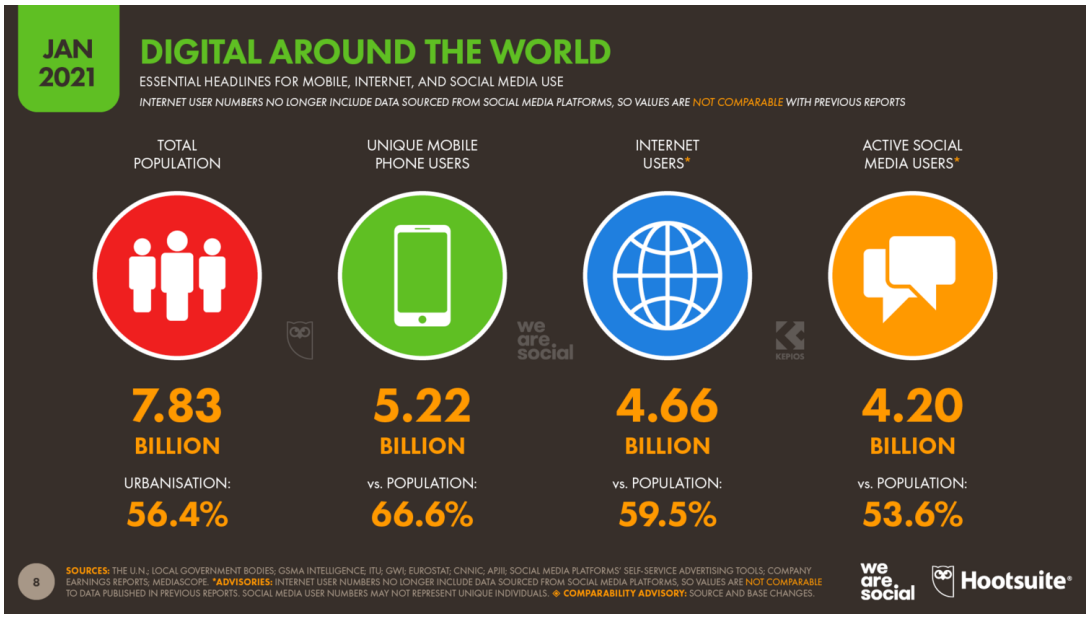

Figura 3: Utilização de conteúdo digital [5]

Este valor corresponde a mais de metade da população de utilizadores ativos no «mundo digital» [5], pelo que se deve ter em consideração no que toca às componentes de segurança na utilização deste tipo de aplicações.

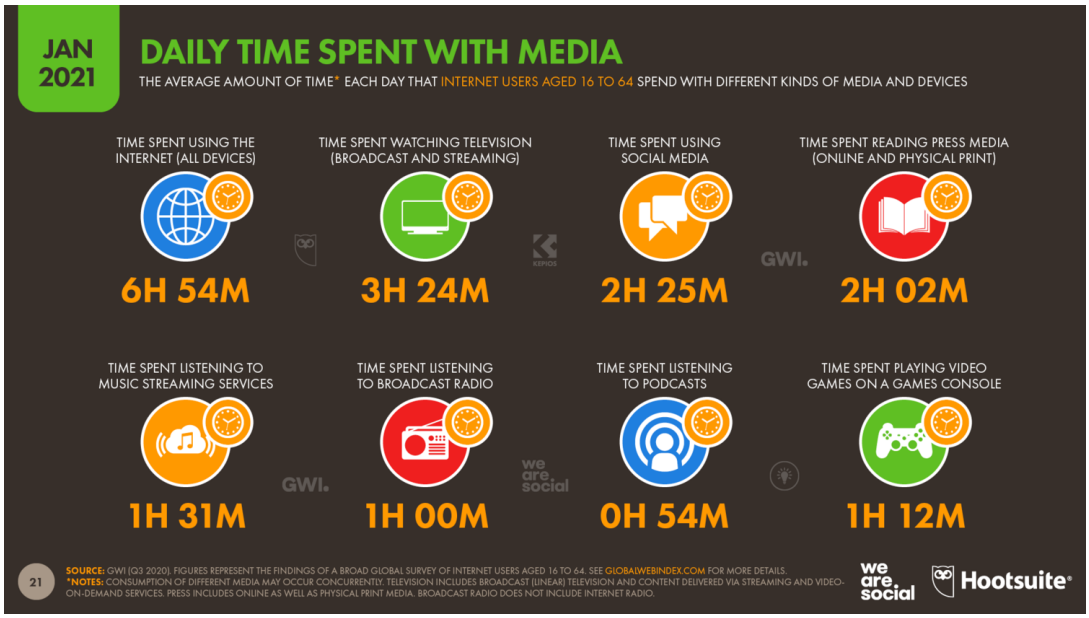

Figura 4: Tempo médio despendido com conteúdo digital [5]

Como se pode verificar na Figura 4, o tempo médio despendido com conteúdo digital em 2021, por utilizadores com idade entre os 16 e os 64 anos, compreende-se maioritariamente em três pontos [5]:

- Utilização de Internet;
- Visualização de programas televisivos (transmitidos pela Internet);
- Utilização de redes sociais.

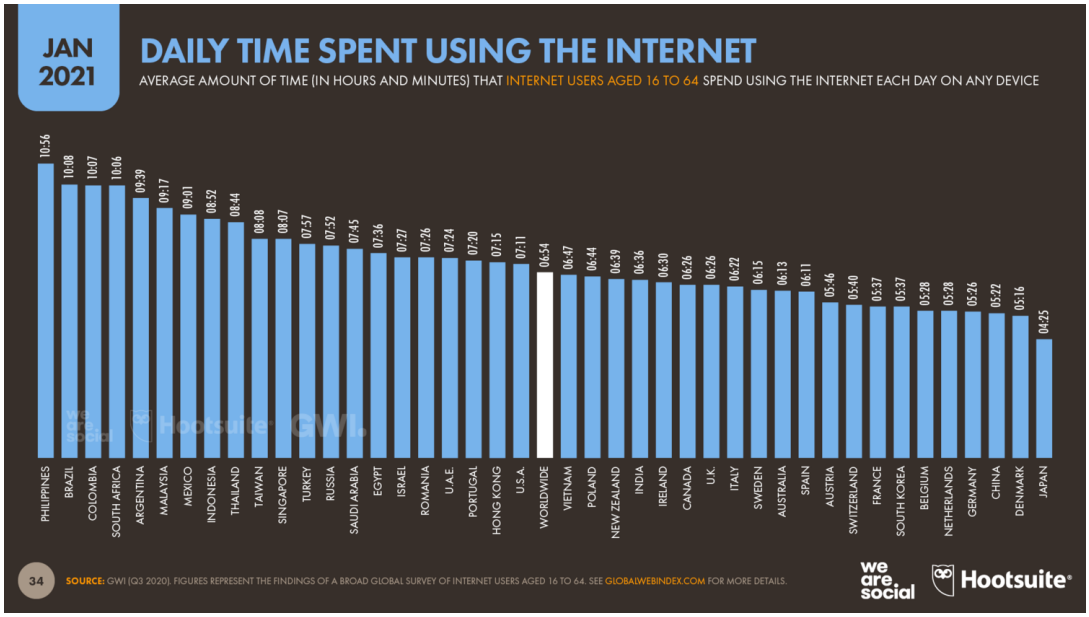

Figura 5: Tempo médio despendido na Internet [5]

Ainda relacionado com o tempo utilizado no «mundo digital», é visível na Figura 5 o tempo médio despendido com utilização de Internet. Pela análise da Figura 5 pode-se verificar o tempo médio despendido em alguns países face à média mundial. Destaca-se que Portugal se encontra acima da média de tempo utilizado na Internet.

Muitas das aplicações das redes sociais têm integradas aplicações de mensagens instantâneas (MI), no entanto, cada aplicação de MI, aplica processos criptográficos de diferentes formas, através da utilização de diferentes protocolos de comunicação, que habilitam segurança aos utilizadores.

Após uma análise destes protocolos, surgiu com maior destaque e despoletou curiosidade, um em particular, o protocolo *Signal* [1].

Efetuada uma análise ao protocolo *Signal* verificou-se que se encontra implementado unicamente em soluções com arquitetura Cliente-Servidor, uma vez que o mesmo tanto requer a utilização do servidor para efetuar a troca de chaves entre os utilizadores, como para o encaminhamento das mensagens.

Assim, pretende-se verificar se é possível aplicar grande parte dos componentes do protocolo *Signal* numa aplicação de *chat* que assenta numa arquitetura PP2P, sem a existência do servidor, mantendo as características do protocolo *Signal*.

1.3 ESTRUTURA

O presente documento é constituído por seis capítulos.

No Capítulo 1, Introdução, é efetuado o enquadramento do trabalho realizado bem como explicados os objetivos e a motivação dos mesmos.

O Capítulo 2, Estado da Arte, relaciona-se com a análise das aplicações de Mensagens Instantâneas que utilizam o protocolo *Signal*, de modo a compreender determinados pontos de segurança abrangidos por estas. Adicionalmente, também é abordada a aplicação desenvolvida no âmbito da UC de Segurança de Redes de Computadores (SRC) por se tratar de uma aplicação de mensagens instantâneas que também aplica processos criptográficos.

No Capítulo 3, Conceitos Relacionados, são abordados termos, conceitos e noções importantes que permitem compreender alguns dos conceitos e desafios necessários para o desenvolvimento da aplicação de *chat* PP2P, tendo em vista o protocolo *Signal* e as suas características.

O Capítulo 4, Arquitetura, destaca a estrutura da aplicação desenvolvida bem como as características criptográficas consideradas como vantagens no âmbito do «protocolo» implementado e da aplicação de *chat* PP2P elaborada.

No Capítulo 5, Implementação, referencia-se a forma como foi desenvolvido o «protocolo» com base nos componentes do protocolo *Signal*, o desenvolvimento da própria aplicação de *chat* PP2P, tendo em conta as suas características. Também aborda sugestões de melhoria à aplicação desenvolvida e ainda as decisões que tiveram de ser tomadas no âmbito da elaboração da aplicação de *chat* PP2P.

Por fim, o Capítulo 6, Conclusões, representa a indicação dos resultados do trabalho desenvolvido bem como são apontadas notas finais à realização da aplicação de *chat* PP2P e ao «protocolo» elaborado.

2

ESTADO DA ARTE

Nesta secção vão ser abordadas e comparadas aplicações de Mensagens Instantâneas. Adicionalmente, também será abordada a aplicação desenvolvida no âmbito da UC Segurança de Redes de Computadores, denominada de *Authority*, que impulsionou a proposta de trabalho que está a ser explorada.

2.1 COMPARAÇÃO DE APLICAÇÕES DE MENSAGENS INSTANTÂNEAS

Pretende-se efetuar uma breve descrição das várias aplicações de forma a contextualizar as mesmas e compará-las em alguns pontos de segurança [6].

	Google Messages	Apple iMessage	Facebook Messenger	Element / Riot	Signal	Microsoft Skype	Telegram	Threema	Viber	Facebook Whatsapp	Amazon Wickr Me	Wire	Session
Overview													
Is the app recommended to secure my messages and attachments?	No	No	No	No	Yes	No	No	Yes	No	No	No	Yes	Yes
Main reasons why the app isn't recommended **Improvements to apps that are recommended** More details	Named as NSA partner in Snowden revelations Makes money from personal data Data not protected, not all data protected No independent & recent code audit and security analysis Closed source	Named as NSA partner in Snowden revelations Data not protected, not all data protected No independent & recent code audit and security analysis Closed source	Named as NSA partner in Snowden revelations Encryption not enabled by default Makes money from personal data Data not protected, not all data protected No independent & recent code audit and security analysis Closed source	No independent & recent code audit and security analysis	Remove the mandatory requirement for users to sign up with a mobile number Provide more comprehensive independent assessments of security/privacy	Named as NSA partner in Snowden revelations Encryption not enabled by default Makes money from personal data Data not protected, not all data protected Closed source	Bespoke cryptography Encryption not enabled by default Data not protected, not all data protected	Make APIs and server code open source Implement perfect forward secrecy at the end-to-end encryption layer Provide more comprehensive independent assessments of security/privacy	Data not protected, not all data protected No independent & recent code audit and security analysis Closed source	Named as NSA partner in Snowden revelations Messages can be read by Facebook if marked as "abusive" Makes money from personal data Data not protected, not all data protected No independent & recent code audit and security analysis Closed source	Former NSA chief Keith Alexander is on Amazon's board of directors Funded by the CIA Recent security audits are not public Closed source	Further limit metadata storage and logging Provide more comprehensive independent assessments of security/privacy	Implement perfect forward secrecy at the end-to-end encryption layer Provide more comprehensive independent assessments of security/privacy

Figura 6: Comparação de aplicações [6]

A Figura 6, representa tabularmente a comparação de alguns pontos de segurança entre várias aplicações, sendo estas as seguintes:

- **Google Messages:**

  Aplicação de Mensagens Instantâneas desenvolvida pela empresa Google.

- **Apple iMessage:**

Aplicação de Mensagens Instantâneas desenvolvida pela empresa Apple, também conhecida por *iMessage*, que permite a comunicação entre várias pessoas quer seja em grupo ou apenas conversa singular.

Esta utiliza o Apple Push Notification Service (APNS) [7] para entregar as mensagens e os anexos aos utilizadores. Quando é efetuado o primeiro registo do utilizador a aplicação cria e armazena o certificado no dispositivo. Para a comunicação das mensagens entre utilizadores, o sistema utiliza o protocolo Transport Layer Security (TLS) [8] para proteger as mensagens APNS [9].

- Facebook Messenger:

  Aplicação de Mensagens Instantâneas desenvolvida pela empresa Facebook. Inicialmente esteve integrada com a aplicação Facebook, tendo sido separada da aplicação anterior facilitando o acesso aos utilizadores, deixando a necessidade de iniciar sessão nesta.

- **Element / Riot:**

  Aplicação de Mensagens Instantâneas desenvolvida pela empresa Element e sustentada no projeto de código livre Matrix.

  Element Matrix Services (EMS) é um serviço de hospedagem considerado robusto e confiável para comunicação segura e rápida em tempo real [10].

  Atualmente, os criadores da aplicação Element são os criadores do projeto Matrix. A empresa permite que a aplicação seja hospedada no servidor relativo ao Matrix, sem custos, ou que seja hospedada pelo utilizador [11].

- **Signal:**

  Aplicação desenvolvida sobre o protocolo Signal, implementado por Signal. Utiliza componentes como X3DH [12] e Double Ratchet [13] de forma a aplicar atributos de segurança ao protocolo.

- **Microsoft Skype:**

  Aplicação de Mensagens Instantâneas desenvolvida pela Microsoft. Permite que sejam criadas conversas singulares ou em grupo de texto ou voz, sendo que para este último tipo de conversa existe a possibilidade de gravação da conversa ou até mesmo adicionar legenda ao vivo [14].

- **Telegram:**

  Aplicação de Mensagens Instantâneas cujo os principais focos são a velocidade e a segurança da comunicação dos seus utilizadores [15]. Esta faz uso do protocolo MTProto [16], desenvolvido pela equipa do Telegram. Este protocolo

é semelhante ao protocolo *Signal*, no entanto não utiliza o X3DH, nem o DR, apesar de aplicar as KDF.

- **Threema:**

  Aplicação de Mensagens Instantâneas que se rege pela segurança e privacidade dos dados dos seus utilizadores [17].

- **Viber:**

  Aplicação de Mensagens Instantâneas, que tal como as restantes, também se preocupa com a segurança e privacidade dos dados dos utilizadores fazendo recurso de métodos criptográficos [18]. Esta permite ao utilizador criar grupos de *chat* em que é possível comunicar por texto, através de mensagens, ou por voz, utilizando as chamadas de voz.

- **Facebook Whatsapp:**

  Aplicação de Mensagens Instantâneas comprada pela empresa Facebook.

- **Amazon Wickr Me:**

  Aplicação de Mensagens Instantâneas cujo foco é a inovação na segurança dos dados dos utilizadores [19]. Esta aplicação é mais direcionada para empresas e organizações dos diversos ramos de negócio.

- **Wire:**

  Aplicação de Mensagens Instantâneas em que a sua visão aponta para a segurança dos dados dos utilizadores [20].

- **Session:**

  Aplicação de Mensagens Instantâneas *open-source* que pretende fornecer segurança e anonimização, através da minimização dos metadados das mensagens enviadas pelos seus utilizadores [21].

A Figura 7 representa a comparação da jurisdição dos dados recolhidos pelas várias aplicações.

	Google Messages	Apple iMessage	Facebook Messenger	Element / Riot	Signal	Microsoft Skype	Telegram	Threema	Viber	Facebook Whatsapp	Amazon Wickr Me	Wire	Session
Details													
Company jurisdiction	USA	USA	USA	UK	USA	USA	USA / UK / Belize / UAE	Switzerland	Luxembourg / Japan	USA	USA	USA / Switzerland	Australia
Infrastructure jurisdiction	Worldwide (rollout on-going, unsure of exact locations, most likely Google Cloud regions)	USA (Ireland and Denmark planned); iMessage runs on AWS and Google Cloud	USA, Sweden (Ireland planned)	UK (and potentially all jurisdictions, given it's a decentralised messaging platform)	USA	USA, the Netherlands, Australia, Brazil, China, Ireland, Hong Kong, and Japan	UK, Singapore, USA, and Finland	Switzerland	USA	USA (unsure of other locations)	USA (unsure of other locations)	EU	Messages: Worldwide (uses de-centralised servers) Attachments: Centralised server in Canada
Implicated in giving customers' data to intelligence agencies?	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	No
Surveillance capability built into the app?	No	No	No	No	No	Yes	No	No	No	No	No	No	No

Figura 7: Comparação de aplicações - jurisdição de dados [6]

É possível verificar que a maior parte das aplicações tem a sua jurisdição fora da zona europeia. Isto implica que a aplicabilidade do Regulamento Geral de Proteção de Dados (RGPD), também conhecido como General Data Protection Regulation (GDPR), fica dificultada uma vez que esta lei diz respeito às pessoas singulares e à sua localização, independentemente da localização geográfica das entidades que tratem os seus dados pessoais [22].

A Figura 8 apresenta a comparação da forma de utilização dos dados por parte das diversas aplicações, bem como o tipo de criptografia aplicada nos processos criptográficos que visam proteger os dados dos utilizadores.

	Google Messages	Apple iMessage	Facebook Messenger	Element / Riot	Signal	Microsoft Skype	Telegram	Threema	Viber	Facebook Whatsapp	Amazon Wickr Me	Wire	Session
Company collects customers' data?	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes	No	No
App collects customers' data?	Yes (Difficult to assess given the app is integrated into Google's greater ecosystem)	Yes (Difficult to assess given the app is integrated into Apple's greater ecosystem)	Health & fitness / purchases / financial info / location / contact info / contacts / user content / search history / browsing history / identifiers / usage data / sensitive info / diagnostics / other data	Contact info / contacts / identifiers / diagnostics / user content (Contact info not sent when using anonymously)	Contact Info	Yes (Information not submitted to Apple Store)	Contact info / contacts / identifiers	Contact info / identifiers / diagnostics (Contact info not sent when using anonymously)	Location / identifiers / purchases / location / contact info / contacts / identifiers / usage data / user content / usage data / diagnostics	Purchases / financial info / location / contact info / contacts / user content / identifiers / usage data / diagnostics	Contact info / usage data / diagnostics (Contact info not sent when using anonymously)	Contact info / identifiers / usage data / diagnostics	No
User data and/or metadata sent to parent company and/or third parties?	Yes	Yes	Yes	No (User data is sent to a third party if a payment is made)	Minimal (mandatory mobile number sent to third party for registration & recovery)	Yes	Yes	No (optional mobile number sent to third party for registration)	Yes	Yes	No (optional mobile number sent to third party for registration)	Yes	No
Is encryption turned on by default?	Yes	Yes	No	Yes	Yes	No	No	Yes	Yes (if device supports it)	Yes (if device supports it)	Yes	Yes	Yes
Cryptographic primitives	Curve25519 / AES-256 / HMAC-SHA256	RSA-1280 (encryption), ECDSA 256 (signing) / AES 128 / SHA-1	Curve25519 / AES-256 / HMAC-SHA256	Curve25519 / AES-256 / HMAC-SHA256	Curve25519 / AES-256 / HMAC-SHA256	RSA-1536 & 2048 / AES 256 / SHA-1	RSA 2048 / AES 256 / SHA-256	Curve25519 256 / XSalsa20 256 / Poly1305-AES 128	Curve25519 256 / Salsa20 128 / HMAC-SHA256	Curve25519 / AES-256 / HMAC-SHA256	ECDH512 / AES-256 / HMAC-SHA256	Curve25519 / ChaCha20 / HMAC-SHA256	X25519 / XSalsa20 256 / Poly1305
Are the app and server completely open source?	No	No	No	Yes (clients Element / Riot, server/API matrix.org)	Yes	No	No (clients and API only)	No (apps only)	No	No	No	Yes	Yes

Figura 8: Comparação de aplicações - utilização de dados e chaves [6]

É possível verificar que apenas cinco empresas, num universo de treze, não recolhem dados dos utilizadores. Adicionalmente, verifica-se que destas cinco empresas, apenas quatro aplicações recolhem dados de identificação do utilizadores para o correto funcionamento das aplicações. Este ponto indica que as empresas detentoras das aplicações podem utilizar os dados que recolhem através das aplicações da forma que entenderem, não respeitando a legislação dos países onde opera, sendo considerado um ponto negativo. Ainda se verifica que as aplicações, das cinco empresas mencionadas anteriormente, duas não enviam os dados recolhidos para terceiras entidades, ainda que das restantes três empresas, duas enviam dados opcionalmente e uma obrigatoriamente. Recordo que nestas três situações, os dados partilhados referem-se a dados de registo de forma a ser possível a recuperação da conta criada na aplicação correspondente.

Para além deste ponto mencionado anteriormente, consegue-se verificar ainda se as aplicações tem os processo criptográficos ligados por definição, sendo que apenas três não o fazem. Posto isto, indica-se como sendo um ponto positivo para as aplicações que por omissão ativam esta propriedade. Ainda referenciando os processos criptográficos também se consegue visualizar quais os tipos de chaves utilizados nestes, destacando-se duas aplicações que utilizam algoritmos do tipo Message Authentication Code (MAC) que já foram comprometidos no passado, retirando a fiabilidade dos processos criptográficos, podendo concluir-se que estamos perante uma situação negativa para estas duas aplicações.

A Figura 9 demonstra a comparação de algumas características de segurança entre as várias aplicações de MI identificadas anteriormente.

	Google Messages	Apple iMessage	Facebook Messenger	Element / Riot	Signal	Microsoft Skype	Telegram	Threema	Viber	Facebook Whatsapp	Amazon Wickr Me	Wire	Session
Can you manually verify contacts' fingerprints?	Yes	No	Yes	Yes	Yes	No	No (session only, does not provide users' fingerprint information)	Yes	Yes	Yes	Yes	Yes	Yes
Directory service could be modified to enable a MITM attack?	N/A, Google Messages uses RCS, which doesn't use a directory service	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Do you get notified if a contact's fingerprint changes?		No		Yes	Yes	No	No (session only, does not provide users' fingerprint information)	Yes	Yes	No (setting turned off by default)	Yes	If contact was previously verified	N/A
Is personal information (mobile number, contact list, etc.) hashed?	N/A, Google Messages uses RCS, which doesn't use a directory service	No	No	Yes	Mostly	No	No	Yes	No	No	Yes	Mostly	N/A
Does the app generate & keep a private key on the device itself?	Yes	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes
Can messages be read by the company?	No	No	Yes	No	No	Yes	Yes	No	No	Yes	No	No	No
Does the app enforce perfect forward secrecy?	Yes	No	Yes	Yes	Yes		No (session keys do change after being used 100 times)	No	Yes	Yes	Yes	Yes	No
Does the app encrypt metadata?	No	No	No		Yes		No	Yes		No	Yes	Mostly	Yes
Does the app use TLS/Noise to encrypt network traffic?	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

Figura 9: Comparação de aplicações - características de segurança [6]

Pode-se verificar que os servidores responsáveis por enviar e receber as mensagens e as chaves, de forma a distribuí-las, podem ser alvo de ataque, viabilizando assim ataques de *Man In The Middle* (MITM).

Adicionalmente, é possível visualizar que todas as aplicações geram e guardam as chaves privadas necessárias para os processos criptográficos no aparelho onde são inicializadas. Isto permite que cada dispositivo contenha o seu conjunto de chaves específico.

Na Figura 9 também é possível verificar que a maioria das empresas não têm acesso às mensagens, apesar de terem acesso aos dados pessoais introduzidos nas aplicações. No entanto, das treze aplicações do universo estudado, quatro entidades conseguem ler as mensagens enviadas pelos utilizadores através das aplicações, sendo uma grave preocupação de privacidade dos seus utilizadores.

Ainda relativamente às características, é realizada a comparação ao atributo *Forward Secrecy* que será abordado na secção 3.3. É possível verificar que apenas afeta quatro aplicações, sendo este um ponto negativo uma vez que pode causar problemas de confidencialidade das mensagens enviadas.

Por fim, é possível analisar quais as empresas e respetivas aplicações que utilizam métodos de cifragem da comunicação entre os utilizadores. Esta metodologia permite dificultar eventuais ataques, pois quer a comunicação quer as próprias mensagens

passam por processos criptográficos, contribuindo de forma positiva no que respeita à segurança da informação dos utilizadores.

Na Figura 10, é possível observar a última parte da tabela comparativa entre as aplicações.

	Google Messages	Apple iMessage	Facebook Messenger	Element / Riot	Signal	Microsoft Skype	Telegram	Threema	Viber	Facebook Whatsapp	Amazon Wickr Me	Wire	Session
Does the app use certificate pinning?		Yes (>=iOS 9.3)			Yes			Yes			Yes	Yes	Yes
Does the app encrypt data on the device? (iOS and Android only)	No	Yes (if passphrase enabled)		Yes	Yes (if passphrase enabled)			iOS: Yes (if passphrase enabled); Android: Yes (if master key set in the app)			iOS: Yes (if passphrase enabled); Android: Yes (unsure of function)	Yes	Yes
Does the app allow a secondary factor of authentication?	No	No	No	No	Yes	No	Yes	Yes	No	Yes	Yes (password for account used)	Yes	Yes
Are messages encrypted when backed up to the cloud?	Yes (>= Android P)	No		Yes	N/A, Signal is excluded from iCloud/iTunes & Android backups			Yes		iOS: Yes Android: Yes	N/A, Wickr is excluded from iCloud/iTunes & Android backups	N/A, Wire is excluded from iCloud/iTunes & Android backups	N/A, Session is excluded from iCloud/iTunes & Android backups
Does the company log timestamps/IP addresses?		Yes	Yes		No	Yes	Yes	No	Yes	Yes	No	Some	No
Have there been a recent code audit and an independent security analysis?	No	No	No	No (Matrix's encryption library reviewed by an independent party)	Yes (October, 2014)	No	Yes (November, 2015)	Yes (October, 2020)	No	No	Yes (August, 2014)	Yes (March, 2018)	Yes (April, 2021)
Is the design well documented?	No	Somewhat	Somewhat	Somewhat	Somewhat	No	Somewhat	Somewhat	Somewhat	Somewhat	Somewhat	Somewhat	Somewhat
Does the app have self-destructing messages?	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes

Figura 10: Comparação de aplicações - características de segurança (continuação) [6]

Na Figura 10 consegue-se observar para as aplicações que disponibilizam informação relativa à utilização de certificados, que todas utilizam certificados de confiança, pelo que apenas aceitam certificados por sua vez gerados por organizações aceites e que dão provas da sua confiança.

Outro ponto que também se verifica é relativo à cifragem dos dados nos dispositivos onde as aplicações estão instaladas. Apenas uma aplicação não utiliza o processo de cifragem dos dados nos dispositivos.

Por fim, após a análise da tabela segregada nas Figuras 6, 7, 8, 9 e 10, é possível retirar algumas conclusões sobre quais as aplicações que consideram mais características de segurança. Assim, verificou-se que as aplicações que utilizam mais políticas de segurança dos dados dos seus utilizadores são o Threema e o Signal.

2.2 TRABALHO PRÉVIO: AUTHORITY

Authority é o nome de uma aplicação desenvolvida por *João Jorge e João Marques* [23] de mensagens instantâneas. Esta mesma aplicação destaca-se pelo uso da arquitetura

Pure Peer-To-Peer e pela forma não controlada de partilha de chaves. Nesta solução, as comunicações estão limitadas ao nível do segmento de rede e o reconhecimento dos *nodes* é realizado através do método de *flooding*.

Para um utilizador poder estabelecer uma futura comunicação, a aplicação define um protocolo para a troca de chaves e para o reconhecimento de *peers*, que pode ser divido nas seguintes fases:

1. **Criação da Estrutura**

   A primeira vez que a aplicação é executada, ocorre a fase designada de *Criação de uma estrutura*, na qual um conjunto de parâmetros do utilizador são automaticamente gerados, incluindo a parte pública de uma chave assimétrica. Este conjunto de parâmetros é denominado por *Contacto* e é usado para a introdução do utilizador na rede. Assim, para o correto funcionamento da aplicação é necessário verificarem-se os seguintes passos:

2. **Introdução**

   Após a criação do *Contacto* do utilizador, a aplicação notifica a sua existência a todos os *peers* na rede através de um *broadcast*.

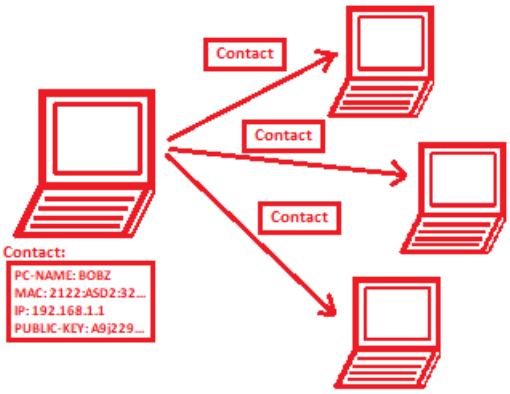

Figura 11: Authority - Introdução

3. **Interpretação**

   Os *nodes* existentes na rede verificam se o *Contacto* é novo e adicionam-o na sua lista de *Contactos*. Caso o *Contacto* já exista na lista de *Contactos*, este é atualizado.

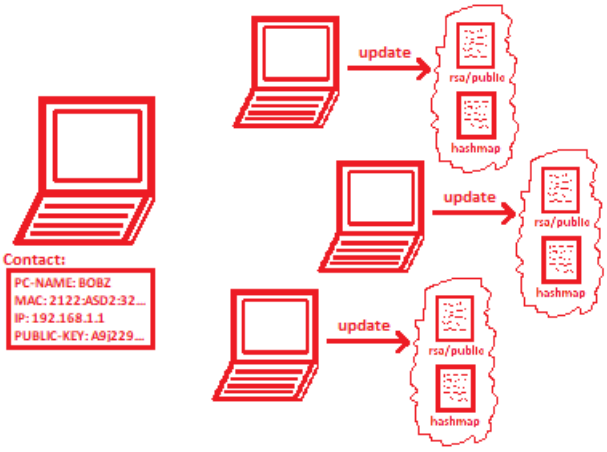

Figura 12: Authority - Interpretação

4. **Integração**

   Após o processamento efectuado anteriormente por cada *node*, estes respondem em Transmission Control Protocol (TCP) **Unicast** com as informações do seu *Contacto*, permitindo, desta forma, que a nova maquina adicione os novos *Contactos* na sua lista e identifique que utilizadores se encontram ativos.

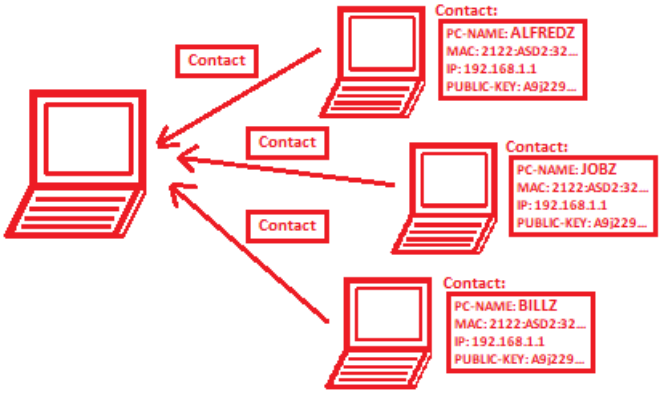

Figura 13: Authority - Integração

Apesar desta aplicação implementar muitas características de segurança semelhantes à aplicação desenvolvida no âmbito deste trabalho, a criação da aplicação *Authority* não é suficiente para mostrar o funcionamento do protocolo *Signal* sem recurso a uma terceira entidade para o funcionamento da aplicação.

3

CONCEITOS RELACIONADOS

Nesta secção serão abordados os conceitos mais relevantes de forma a contextualizar o trabalho desenvolvido, nomeadamente os conceitos de P2P e o protocolo *Signal*, utilizado como base do desenvolvimento do «protocolo» implementado. Por fim, também será abordada uma análise criptográfica a aplicações de Mensagens Instantâneas que permite perceber alguns conceitos de segurança aplicados na aplicação desenvolvida.

3.1 PEER-TO-PEER (P2P)

A arquitetura P2P caracteriza-se pela igualdade dos serviços/funcionalidades disponibilizadas por cada interveniente (*node*), isto é, cada nó numa rede P2P tem as mesmas características.

A imagem seguinte (Figura 14) representa um exemplo de uma arquitetura P2P.

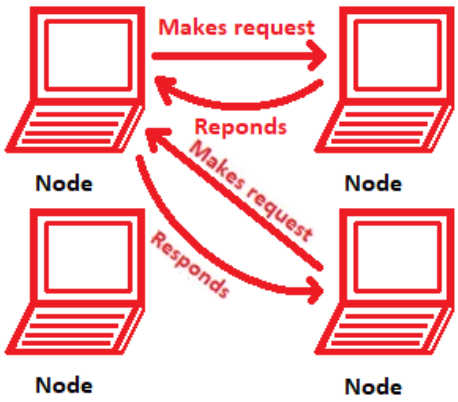

Figura 14: Arquitetura P2P

É possível verificar que cada elemento da rede (*node*) efetua e responde a pedidos realizados por outros elementos da mesma rede, convergindo assim com a definição mencionada anteriormente.

3.2 PROTOCOLO SIGNAL

O protocolo *Signal*, desenvolvido por *Moxie Marlinspike e Trevor Perrin* [24], é um protocolo de baixa latência e assíncrono que permite a comunicação segura entre várias entidades. Neste protocolo são abordados vários mecanismos de segurança, sendo os mais relevantes:

- *X3DH* [12]:

  Protocolo usado na partilha das chaves entre os utilizadores;

- *Double Ratchet* [13]:

  Protocolo utilizado na troca de mensagens para aplicar segurança a este processo, fazendo uso das características *forward secrecy* e *future secrecy*.

3.2.1 *X3DH*

O *Extended Triple Diffe-Hellman (X3DH)* é um protocolo de *key-agreement* que estabelece uma chave secreta partilhada entre duas entidades. Deste modo, o objetivo assenta na promoção das propriedades criptográficas de *forward secrecy* e *cryptographic deniability*, previamente abordadas na secção 3.3. Neste protocolo, as entidades envolventes autenticam-se mutuamente através das suas chaves públicas. O protocolo *X3DH*, pode ser descrito pelas seguintes fases:

3.2.1.1 *Inicialização*

Na primeira fase de inicialização, no qual o utilizador por simplificação Bob abre a aplicação MI pela primeira vez, são criadas combinações de chaves assimétricas de curvas elípticas:

- **Identity Key - IKb**: chave única identificativa do utilizador Bob;
- **Signed Prekey - SPKb**: chave que garante que o utilizador controla a sua chave pública, regularmente actualizada (semanas ou meses);
- **Prekey signature - Sig(IKb, Encode(SPKb))**: chave regularmente actualizada;
- **Lista de One-Time prekeys - (OPKb1, OPKb2, ...)**: lista com chaves de uso único, que podem ser adicionadas conforme a necessidade. Estas chaves são opcionais, não sendo necessário o seu acréscimo no processo.

Após a criação destas chaves, a aplicação IM envia-as para o servidor, como demonstrado na Figura 15, com vista a permitir o estabelecimento assíncrono de futuras comunicações.

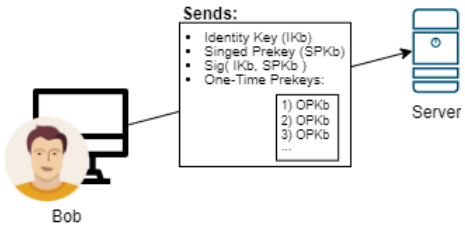

Figura 15: Protocol X3DH - Inicialização

3.2.1.2 *Obtenção do Contacto*

Uma segunda entidade, por simplificação denominada por Alice, pretende estabelecer um canal de comunicação com o Bob, mas existem determinados desafios que, segundo Pound [25], podem surgir:

- O Bob encontrar-se *offline* (assíncrono);
- O servidor ser o único que conhece a localização do Bob, a título de exemplo: endereço IP, telefone (abordado na implementação do *Authority* secção 2.2);

De forma a resolver tais problemas, a implementação do protocolo *Signal* recorre ao uso de uma terceira entidade-servidor. Este é responsável por enviar um pacote denominado por *Prekey Bundle*, que contém o conjunto de chaves previamente enviadas para a Alice, tal como demonstrado na Figura 16. Caso não existam *One-Time Prekeys* disponíveis, o servidor cria e envia um *Prekey Bundle* sem esta chave.

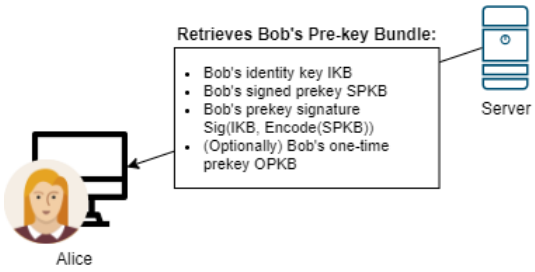

Figura 16: Protocol X3DH - Obtenção do contacto do Bob

Após o envio das chaves, o servidor é responsável por eliminar a *One-Time Prekey* enviada na *Prekey Bundle*.

3.2.1.3 *Criação da chave Ephemeral*

Ao receber a *Prekey Bundle*, a Alice verifica se a assinatura é válida através da equação 1:

$$\mathsf{Sig}\,(\mathsf{IKB}, \mathsf{Encode}\,(\mathsf{SPKB})) \tag{1}$$

Após o cálculo da assinatura, a Alice compara o valor enviado na *Prekey Bundle* com o calculado, caso os valores não coincidem a operação é abortada. De seguida, a Alice gera um par de chaves *ephemeral*, tal como ilustrado na seguinte Figura 17.

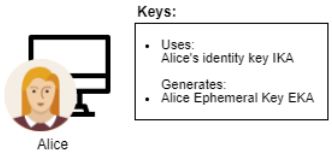

Figura 17: Protocol X3DH - Alice gera uma Ephemeral key

3.2.1.4 *Troca de Mensagens*

Neste momento, a Alice dispõem de todos os elementos necessários para a criação de um canal seguro com o Bob, através do algoritmo *Diffie-Hellman*, que tem a particularidade de permitir a troca de chaves seguras num ambiente inseguro. É possível derivar as chaves que serão usadas para criar uma chave secreta, como representado na Figura 18:

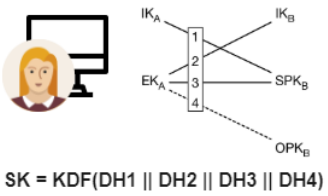

Figura 18: Protocol X3DH - Alice aplica o Triple Diffie-Hellman

No processo previamente ilustrado, a Alice aplica quatro *Diffie-Hellman*, nomeadamente:

$$\begin{aligned} \mathsf{DH1} &= \mathsf{DH}(\mathsf{IKA}, \mathsf{SPKB}) \\ \mathsf{DH2} &= \mathsf{DH}(\mathsf{EKA}, \mathsf{IKB}) \\ \mathsf{DH3} &= \mathsf{DH}(\mathsf{EKA}, \mathsf{SPKA}) \\ \mathsf{DH4} &= \mathsf{DH}(\mathsf{EKA}, \mathsf{OPKB}) \end{aligned} \tag{2}$$

Os dois primeiros *Diffie-Hellman* realizados, *DH1* e *DH2*, são aplicados de forma a garantir a autenticação de ambas as partes, ou seja, do Bob e da Alice. Enquanto que, o *DH3* e o opcional *DH4* tem o intuito de permitir *forward secrecy*.

Após a realização do *Triple Diffie-Hellman*, todos os membros resultantes são concatenados e aplicados a uma *Key Derivation Function (KDF)*, como demonstrado na Figura 18.

Adicionalmente, a Alice calcula o código *Associated Data (AD)*, demonstrado na Equação 3, que contém a junção das chaves *IK* da Alice e do Bob e que verifica a existência de um canal de comunicação entre ambas as partes.

$$\mathsf{AD} = \mathsf{Encode}(\mathsf{IKA}) \,|\mathsf{Encode}(\mathsf{IKB}) \tag{3}$$

Por fim, a Alice envia ao utilizador Bob uma mensagem contendo as seguintes chaves:

- **Identity Key - IKa**: chave única identificativa da Alice;
- **Ephemeral Key - EKa:** chave assimétrica destrutível após uso;
- **One-Time preykey - OPKb**: one-time prekey do Bob que a Alice usou;
- **Encrypt(SK, AD)**: através de um algoritmo Authenticated Encryption with Associated Data (AEAD) encriptar o valor *AD* com a chave *SK* gerada.

3.2.2 *Double Ratchet*

O algoritmo *Double Ratchet* (DR) é utilizado no protocolo *Signal*, para efetuar a troca de mensagens cifradas entre duas entidades, tendo como base um segredo partilhado por ambas. Normalmente, esta troca é concretizada pelo protocolo de troca de chaves *X3DH* [13].

Segundo [13], na base do DR estão as cadeias *Key Derivation Function* (KDF), que tem as seguintes propriedades:

- **Resiliência** - As chaves geradas pelo KDF são aleatórias, não sendo possível obtê-las mesmo sabendo o *input*;
- ***Forward secrecy*** - As chaves geradas são diferentes, garantindo que quando esta for quebrada apenas as mensagens cifradas com esta chave estarão comprometidas;
- ***Break-in recovery*** - As chaves futuras serão diferentes, mesmo que um atacante obtenha uma chave. Isto acontece devido à entropia causada pelos parâmetros de *input*.

Numa conversação, entre duas entidades, em que é utilizado o DR, cada uma cria três cadeias KDF, *Diffie-Hellman Chain* (DHC), *Sending Chain* (SC) e *Receiving Chain* (RC) [13] [26].

O nome de *Double Ratchet* é atribuído através da existência de um *ratchet* para a chave simétrica (SC/RC) e outro para a iteração do algoritmo *Diffie-Hellman* (DH). Estes *ratchets* são ativados quando:

- Uma mensagem é enviada ou recebida. Assim, o SR ou RC efetua uma iteração de modo a derivar a chave para a cifragem ou decifragem;
- Uma nova chave pública é recebida, através do *header* da mensagem. Desta forma, o DHR efetua uma iteração para realizar a troca da chave do *ratchet*, de forma a permitir a decifragem da mensagem seguinte.

3.3 ANÁLISE CRIPTOGRÁFICA DE APLICAÇÕES DE MENSAGENS INSTANTÂNEAS

As Mensagens Instantâneas são utilizadas, segundo [27], para a comunicação entre duas ou mais entidades. Estes sistemas de mensagem, ao contrário dos seus antecessores (*SMS*), permitem suportar sem quaisquer custos funcionalidades como a troca de mensagens e transferência de ficheiros (imagens, vídeos, etc).

Muitos dos protocolos de MI encontram-se centralizados de tal forma, que o utilizador não tem capacidade de selecionar o servidor de confiança responsável pela comunicação, já que este é disponibilizado pela entidade prestadora do serviço.

Dependendo do protocolo, as aplicações de MI podem suportar conversas em grupo administrados por utilizadores nomeados ou por todos os membros integrantes. No

estudo efetuado por Paul Rosler [27], foram analisadas as propriedades criptográficas nas conversações em grupo dos protocolos de MI mais comuns. Contudo, a maioria destas propriedades não se focam apenas em conversas em grupo, podendo ser aplicadas num cenário de, apenas, dois participantes, sendo estas:

- **Confidentiality**:
  - **End-to-end Confidentiality**:

    Este ponto indica que nenhuma mensagem enviada pode ser obtida por uma entidade não autorizada.
  - **Perfect Forward Secrecy**:

    Caso a sessão seja comprometida assegurar a confidencialidade das mensagens previamente enviadas.
  - **Future Secrecy**:

    Na situação em que uma sessão esteja comprometida a confidencialidade das futuras mensagens é restabelecida.
- **Integrity**:
  - **Message Authentication**:

    Característica que garante que uma mensagem é, de facto, enviada pela entidade emissora.
  - **No Creation**:

    Este ponto indica que apenas pessoas autorizadas podem comunicar numa chamada.
  - **No Duplication**:

    Esta característica representa que a mesma ação não pode ser efectuada em simultâneo pelos mesmos elementos, sendo que esta só pode ser executada uma vez por cada membro do grupo.
  - **Traceable Delivery**:

    Uma característica mais especifica de *chats* de grupo que representa as notificações das mensagens enviadas, isto é, a cada mensagem enviada, os membros presentes no grupo, podendo ser dois ou mais elementos, são notificados de por uma ação efetuado por outro elemento do grupo.
    * **Weak FIFO Order**:

Garantir que uma acção só é executada se não houver acções pendentes.

* **Weak Causal Order**:

  Garantir que uma acção só pode ser criada quando todas as acções estiverem devidamente processadas.

- **Closeness**

  Esta característica garante que apenas os utilizadores autorizados tenham acesso à comunicação nos *chats* de grupo, ou seja, *closed room*.

Neste trabalho, foram definidos os vários ataques que um sistema de MI pode sofrer e que, por sua vez, podem ser usados para a avaliação e a base da solução a ser desenvolvida. Estes ataques podem, portanto, enquadrar-se nos seguintes tipos:

- **Malicious User**:

  Quando um utilizador mal intencionado que pode explorar as falhas nas especificações do protocolo.

- **Network Attacker**:

  Isto é, um atacante com controlo total sobre a comunicação na rede, podendo aceder e modificar o tráfico não protegido.

- **Malicious Server**:

  Ou seja, um atacante com controlo sobre o servidor central e que pode realizar ataques na camada de transporte.

- **Long-term Secret Compromise**:

  Quando um atacante é capaz de comprometer um utilizador, tanto durante como depois da execução do protocolo, de forma a obter as chaves secretas de longo termo.

- **Session State Compromise**:

  Isto é, um atacante que obteve a sessão de um utilizador num estado intermédio da execução do protocolo.

Pretende-se que o presente trabalho permita juntar os dois conceitos, P2P e o «protocolo» *Signal* de forma a criar um sistema que garanta as propriedades do protocolo *Signal* sem o requisito da existência de um servidor intermediário na comunicação entre entidades.

4

ARQUITETURA

Neste capitulo pretende-se descrever de forma global a arquitetura da rede bem como a estrutura da aplicação de *chat* P2P.

A arquitetura da aplicação de *chat* é uma arquitetura PP2P, isto é, qualquer participante na rede P2P (nó), tanto representa um servidor como um cliente sem que exista outro nó para «regular» os pedidos e respostas enviados entre os vários participantes.

A figura 19 representa esquematicamente a arquitetura da rede P2P em cuja aplicação de *chat* assenta.

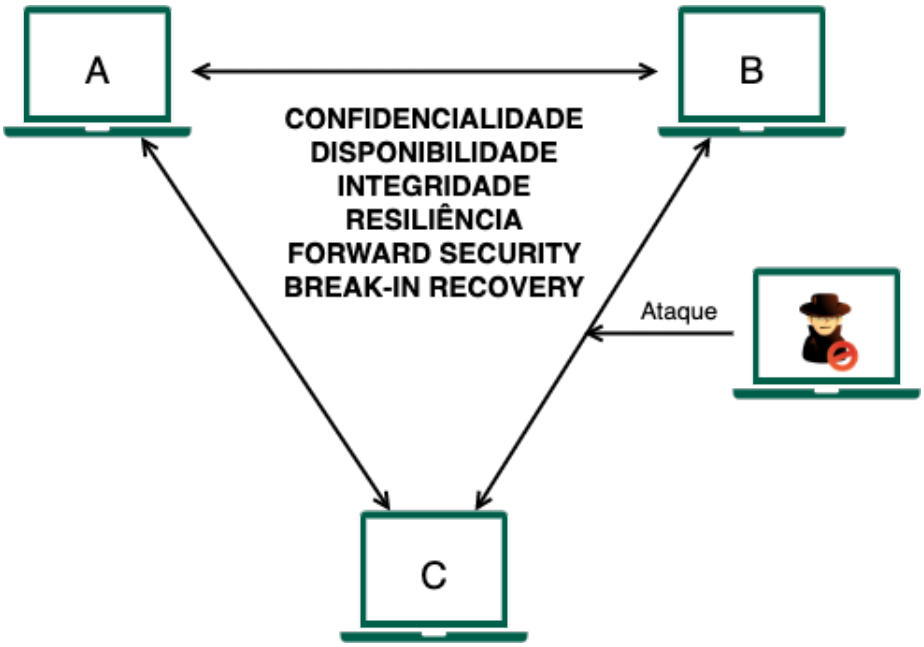

Figura 19: Arquitetura da rede

Na Figura 19 é possível verificar, para além da arquitetura da rede, uma tentativa de ataque à rede entre as entidades B e C. Um atacante que tente obter o pacote que contem a mensagem enviada entre as duas entidades, apenas obtém a mensagem cifrada. Essa mensagem terá sido cifrada com a utilização de componentes semelhantes aos do protocolo Signal, pelo que, será bastante difícil de decifrar uma vez que

todas as mensagens cifradas pelo protocolo proposto seguem algumas características criptográficas.

Tal como mencionado anteriormente, a aplicação de *chat* foi desenvolvida com a aplicação de um «protocolo» de comunicação que utiliza componentes do protocolo Signal, assim, as características criptográficas mencionadas anteriormente que permitem aumentar a segurança dos utilizadores são as seguintes:

- **Confidencialidade**:

  Esta característica garante aos utilizadores da aplicação de *chat* que a informação enviada entre as várias entidades é apenas do conhecimento das entidades envolvidas na mensagem [28].

- **Integridade**:

  Esta característica garante que as mensagens enviadas entre as entidades participantes são de facto originadas da entidade que as enviou, levando aos atributos de não repudio e autenticação, e ainda permite afirmar que estas não foram modificadas ou destruídas por possíveis ataques [29].

  A aplicação de *chat* garante a integridade dos dados através da utilização de processos criptográficos que se servem de chaves assimétricas para cifrar as mensagens enviadas pelas entidades envolvidas no processo de comunicação. Este processo está mais detalhado no Capitulo 5, na secção 5.4.

- **Disponibilidade**:

  Este atributo permite aos utilizadores terem a aplicação disponível a qualquer altura, em qualquer lugar [30].

  No caso da aplicação de *chat*, uma vez que a arquitetura é P2P, esta estará disponível em qualquer momento, em qualquer lugar.

- **Resiliência**:

  Este atributo permite que a aplicação de *chat* mantenha as suas funcionalidades face a ataques [31].

  Para a situação da aplicação de *chat*, caso um atacante controle os *inputs* para as KDF, não conseguirá obter a chave utilizada para a cifragem de uma determinada mensagem [13].

  Isto acontece pois é gerada uma nova chave a cada mensagem enviada, tal como referido no Capitulo 5, na secção 5.4.4.

- ***Forward security***:

É uma característica que garante a segurança de mensagens anteriores, inviabilizando a quebra de segurança.

No caso de um atacante obter uma chave utilizada na cifragem de uma das mensagens enviada entre as entidades, não conseguirá obter as mensagens enviadas anteriormente uma vez que a chave que foi utilizada no processo criptográfico anterior não é a mesma, e não é possível utilizar a KDF para gerar chave anterior pois esta não permite o processo inverso [13].

Esta característica está assegurada pela utilização do componente *Double Ratchet*, que pode ser consultado com maior detalhe no Capítulo 5 na secção 5·4·4.

- ***Break-in recovery***:

  Atributo que garante a segurança das mensagens seguintes enviadas entre as entidades.

  Este ponto permite que em, caso de ataque e de obtenção da chave secreta gerada para o processo, não seja possível a obtenção da próxima chave a ser utilizada na função criptográfica KDF uma vez que sem o *input* para a função não é possível obter a chave seguinte. [13]

  Esta característica é aplicada através da utilização da componente *Double Ratchet* do protocolo Signal como é visível no Capítulo 5 na secção 5·4·4.

5

IMPLEMENTAÇÃO

Com este trabalho pretende-se verificar a possibilidade da utilização do protocolo Signal numa aplicação de *chat* de arquitetura Pure Peer-To-Peer. Pretende-se que este *chat* seja desenvolvido sobre a *framework ElectronJS* visto ser uma tecnologia recente que permite que as aplicações sejam *Cross-Platform*, isto é, podem ser utilizadas em modo aplicação ou via *browser* e em vários sistemas operativos, como macOS, Windows e Linux.

5.1 METODOLOGIA APLICADA

A metodologia seguida para o desenvolvimento do projeto foi em parte Scrum [32], uma vez que não foram cumpridas todas as cerimónias base descritas no manifesto. Numa primeira fase, foram realizadas reuniões quinzenais, sendo que estas foram adaptadas para reuniões semanais de modo a permitir um acompanhamento mais próximo do trabalho desenvolvido, bem como a esclarecer ideias para a implementação do *chat* quer ao nível do protocolo, quer ao nível do design gráfico. Nestas reuniões também foram discutidas ideias relativamente ao relatório do desenvolvimento do *chat*.

O trabalho foi dividido em duas partes bem definidas; a componente prática, a prova de conceito, e a componente teórica, o relatório. Dadas as circunstâncias laborais, o plano não foi cumprido à risca, no sentido em que estavam definidos prazos mais promissores que acabaram por ter de ser ajustados. Apesar dos contratempos, as reuniões quinzenais e semanais foram importantes no desenrolar do trabalho, pois permitiram que existisse pressão constante para o desenvolvimento do trabalho, mesmo em contexto pós-laboral, levando a que não existisse tanta perda de foco na elaboração do trabalho, sobretudo na componente teórica.

5.2 EXPLORAÇÃO TÉCNICA

Nesta secção pretende-se abordar os módulos/bibliotecas externas utilizadas para ajudar a desenvolver a ideia original.

No arranque da aplicação, esta cria uma entidade associada ao utilizador. Esta entidade é fundamental para a comunicação com os restantes nós da rede. Quando um utilizador (nó) pretende contactar com outro, numa fase inicial, deve introduzir o endereço IP do nó que pretende contactar de forma a criar uma ligação entre ambos. Esta despoleta um pedido de um conjunto de chaves do outro utilizador que permitem a cifragem da informação que será transmitida entre os intervenientes. Este conjunto de chaves é conhecido por *bundle* e é composto por chaves enviadas pela entidade que se pretende comunicar, sendo fundamental para os processos criptográficos das mensagens instantâneas.

5.2.1 *Módulos/Bibliotecas utilizados*

Como referido anteriormente, foi utilizado para este projeto a *framework* ElectronJS. Esta acenta sobre as tecnologias nodeJS e JavaScript.

Posto isto, foi elaborada uma pesquisa sobre módulos para nodeJS que permitissem a implementação do protocolo *Signal*. Começamos por verificar a existência de alguma biblioteca/módulo dos criadores do protocolo *Signal*. Durante esta pesquisa, verificámos a existência de um módulo para *javascript*, que pareceu satisfazer as necessidades para a implementação do protocolo. No entanto, após vários testes, não foi possível implementar, uma vez que existiam bastantes dependências que teriam de ser acrescentadas à medida do desenvolvimento do código. Estes módulos são visíveis na Tabela 1.

Tabela 1: Módulos da aplicação *Signal*

Componente Signal	**libsignal**	**rawr-x3dh**	**sodium-plus**
Eliptic-Curve	X	X	X
X3DH	X	X	X
Double Ratchet	X		X

O módulo libsignal [33] foi desenvolvido pelos criadores do protocolo Signal (Moxie Marlinspike e Trevor Perrin). Este módulo está disponivel para três tecnologias diferentes, sendo que uma delas é JavaScript. Posto isto, optámos por seguir com o módulo. Durante a implementação, verificámos problemas com as dependências, uma vez que a cada erro encontrado e corrigido aparecia logo outro com a mesma causa. Com este cenário em mão, optámos por tentar encontrar uma nova alternativa de pesquisar outro módulo/biblioteca que permitisse efetuar, separadamente, as diversas etapas do protocolo *Signal*.

O rawr-x3dh [34] é um módulo que permite a realização do X3DH. Esta solução tem o objetivo de efetuar a troca de chaves de forma a criar um segredo conhecido por ambas a entidades, segredo esse que é utilizado nos métodos de cifragem com maior velocidade no que toca à entrega da mensagem no destino. Assim, este módulo facilita a implementação desta componente do *Signal* pois após a devida configuração do servidor, apenas requer a utilização de métodos no código.

O sodium-plus [35] é um módulo que permite, através de um conjunto de métodos, a aplicação de algoritmos criptográficos, quer de geração de chaves utilizadas nas várias alternativas de cifragem da informação, quer na própria utilização dos algoritmos para cifrar os dados.

Para efetuar a ligação entre os vários *peers* na rede foi utilizado a *framework* socket.io

A *framework* socket.io [36] assenta sobre a linguagem JavaScript e permite a conexão dos utilizadores da rede (nós) para o envio e receção das mensagens através da criação do servidor e clientes.

O servidor fica responsável pela receção dos pedidos efetuados pelos clientes, enquanto que os clientes ficam responsáveis pelo envio dos pedidos. Na aplicação de *chat* desenvolvida, os pedidos enviados pela rede correspondem as mensagens cifradas.

Para a componente gráfica do *chat*, foi utilizada a *framework* alpineJS em conjunto com Bootstrap 5.

- AlpineJS [37] é uma *framework* que acenta na linguagem JavaScript que permite a criação de componentes de código de modo a acelerar o desenvolvimento do *front-end* da página *web*.
- Bootstrap5 [38] é uma *framework open-source* também assente na linguagem JavaScript utilizada na personalização de páginas *web*.

5.2.2 *Desafios identificados*

Apesar de se ter comprovado a viabilidade deste protocolo em determinadas condições, identificaram-se alguns pontos de melhoria face à aplicação de chat criada.

Considerou-se a questão relativamente à sincronização dos *ratchets* pois uma vez que não existe um servidor intermédio, existe dificuldade em garantir o sincronismo dos *ratchets* caso uma das entidades não receba as mensagens. Existe a hipótese de utilizar um dos nós da rede para deixar a mensagem cifrada caso a entidade para o qual se pretende enviar a mensagem não esteja disponível, permitindo assim a receção das mensagens. Esta solução, no entanto, exige que estejam sempre três nós na rede de forma a que quando um falhar, exista outro disponível para guardar a mensagem até à próxima ligação de rede do nó que deve receber a mensagem.

Outro ponto identificado refere-se ao armazenamento das chaves. De momento, estas são armazenadas em ficheiro no dispositivo da entidade. De forma a facilitar o acesso entre vários dispositivos sugere-se a utilização de bases de dados descentralizadas, guardando de forma cifrada as chaves geradas nos processos de criação das entidades, bem como dos *bundles* recebidos e criados para outras entidades. A aplicação de bases de dados descentralizadas pronuncia uma alteração ligeira da arquitetura da aplicação uma vez que existe a necessidade de um servidor para regular as bases de dados disponíveis às entidades da aplicação de *chat*. Este servidor apenas asseguraria que as bases de dados fiquem sincronizadas.

5.3 APLICAÇÃO DE CHAT

No âmbito deste projeto foi desenvolvido um *chat* P2P com a utilização de componentes aplicados no protocolo *Signal*, sendo estes a utilização de chaves de curva elíptica, o protocolo X3DH e *Double Ratchet* [1].

Assim, implementou-se um sistema distribuído de chaves para efetuar a troca de chaves entre os utilizadores (nós) da rede.

Cada nó da rede é traduzido para uma entidade, sendo esta a detentora das chaves geradas para os processos criptográficos, bem como a entidade armazenadora das chaves públicas que vai recebendo das restantes entidades da rede P2P.

As entidades são compostas por chaves, sendo que estas podem ter origem em dois pontos distintos:

- Chaves criadas pela própria entidade;
- Chaves criadas por outras entidades.

As chaves criadas pela própria entidade são chaves que são instanciadas e inicializadas com o arranque da aplicação de *chat*, uma vez que assim que esta arranca, os vários componentes são inicializados.

As chaves criadas por outra entidades são chaves que são partilhadas na forma de um *bundle* (conjunto de chaves). Para a nossa entidade ter acesso ao *bundle* de chaves de outras entidades tem primeiramente que efetuar um pedido. Este é estabelecido com o pedido de comunicação para outra entidade.

Para duas entidades (A e B) comunicarem é necessário que seja efetuado um pedido de comunicação da entidade A para a entidade B, sendo que este pedido, de momento, é realizado na interface gráfica da aplicação de *chat*. O utilizador terá de colocar o endereço da outra entidade para ser efetuado o pedido de comunicação e por consequência, o pedido de *bundle* de chaves, de modo a suportar os processos criptográficos no envio e receção de mensagens.

No momento de efetuar a cifragem da mensagem a enviar entre as duas entidades, as chaves são utilizadas, primeiro aplicando o Triple Diffie-Hellman (X3DH) (no caso quatro vezes devido à chave do tipo One Time Password (OTP)), depois aplicando o Double Ratchet (DR), sendo que um deles é o de envio das mensagens com a atualização do segredo calculado em cada mensagem cifrada, o outro o de receção das mensagens que necessita de ser atualizado com o calculo da SK para a mensagem recebida. Cada um destes *ratchets* tem de estar sincronizado para o correto funcionamento do protocolo.

5.3.1 *Electron JS*

ElectronJS [39] é uma *framework* para o desenvolvimento de aplicações para *desktop* com base em JavaScript, HTML e CSS. Esta integra componentes como Chromium e Node.js de forma a permitir que as aplicações criadas sejam multi-plataforma, isto é, permite que seja utilizada por utilizadores de Windows, macOS e Linux [39].

De forma a evidenciar uma visão mais integra da *framework*, é importante clarificar estes dois componentes.

- **Chromium**:

O componente Chromium está integrado nos projetos Chromium, sendo definido como um navegador *open-source* cujos objetivos são a segurança, a estabilidade e a velocidade [40].

- **Node.js**:

  Node.js é um componente desenvolvido sobre o motor de JavaScript V8 desenvolvido pela Google que assenta sobre a linguagem JavaScript [41].

A *framework* Electron JS, tal como os *browsers* [42] utiliza um modelo de multi processos. O esquema seguinte, representa o modelo de processos da *framework* Electron, que é bastante semelhante ao modelo de processos do navegador *web* Google Chrome.

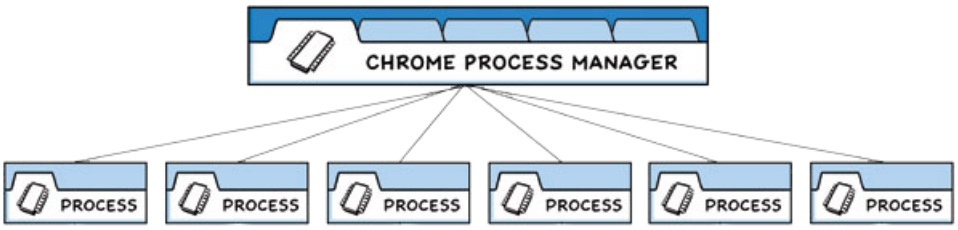

Figura 20: Exemplo do modelo de processos do Chrome [42]

A segregação dos processos, para além de reduzir o nível de processamento de cada separador/processo, em caso de problemas de performance de um dos processos, como por exemplo lentidão, este não afeta os restantes processos. Esta segmentação faz com que muitas das vezes, estes temas sejam transparentes para o utilizador.

- **Renderer Process**

  O Renderer Process é o processo ligado a cada janela criada na aplicação, cuja sua função é processar o conteúdo *web* da página [42].

  O Renderer Process é responsável pela renderização do conteúdo *web* da página, isto é, apresenta visualmente os diversos componentes que compõem a aplicação de *chat*.

  Adicionalmente envia as mensagens internamente para o Main Process através do Inter-Process Communication (IPC), de forma a permitir este aplicar a cifragem às mensagens.

  Para a componente gráfica do *chat*, foi utilizada a *framework* alpineJS [37] em conjunto com Bootstrap 5 [38].

  – **AlpineJS** :

É uma *framework* que permite de forma simplificada a ligação entre o código JavaScript e página HTML. Com recurso a este módulo é possível utilizar componentes customizadas pela *framework* para o funcionamento da aplicação de *chat*.

- **Bootstrap 5** :

  Framework utilizada para o *design* gráfico da aplicação de *chat*. Este modulo compõe um conjunto de componentes que simplifica a construção da interface gráfica da aplicação de *chat*.

- **Main Process**

  O Main Process é o processo responsável pelo controlo dos processos associados à *framework*. Este comunica com os diversos Renderer Processes [42].

  É utilizado na aplicação para efetuar as ligações entre os *peers*, bem como para aplicar os métodos criptográficos sobre as mensagens a ser enviadas entre os nós.

- **Comunicação entre processos** Para a comunicação entre processos, a *framework* electronJS contempla o IPC. Este método permite a comunicação entre o Renderer Process e o Main Process.

  Um exemplo prático da comunicação entre os processos é o envio e receção de uma mensagem. Para o envio da mensagem, esta começa a ser elaborada no *front end* da aplicação pelo utilizador. De seguida é cifrada e enviada através do IPC do Renderer Process para o Main Process, *back end* da aplicação.

  O Main Process, tal como descrito anteriormente, fica responsável por enviar a mensagem até ao cliente destino.

5.4 COMPONENTES CRIPTOGRÁFICOS

Esta secção destina-se a identificar as componentes criptográficas utilizadas no desenvolvimento da aplicação de *chat*, de forma a implementar o protocolo *Signal*.

5.4.1 *Sodium-Plus*

O Sodium-Plus [35] é uma biblioteca desenvolvida sobre a linguagem JavaScript que utiliza um módulo denominado de libsodium [43], que por sua vez, permite a

implementação de algoritmos de cifragem, decifragem, assinaturas, *passwords*, *hashs*, entre outros.

Este módulo baseia-se na criação de métodos de implementação dos algoritmos. Assim, facilita a instanciação deste métodos de cifragem, bem como a sua utilização.

O objetivo principal deste módulo é ser uma ferramenta criptográfica que permite efetuar todas as operações base.

A implementação do protocolo Signal teve por base o módulo Sodium-Plus. Este módulo permitiu efetuar a geração de chaves a serem distribuidas entre os utilizadores da aplicação de *chat*.

5.4.2 *Curvas Elipticas ED25519 e X25519*

Para o funcionamento da aplicação foi necesário a criação das chaves do tipo ED25519 e X25519.

- **ED25519**:

  O algoritmo de cifragem Edwards-curve Digital Signature Algorithm (EdDSA) é uma variante dos algoritmos Edwards-curve. É um algoritmo de cifragem assimétrica, uma vez que é utiliza pares de chaves, publica e privada. Foi introduzido em 2015 [44].

  Este algoritmo pretende gerar e verificar assinaturas de chave publica de forma rápida e com maior robustez face a ataques externos.

- **X25519**:

  O algoritmos de cifragem X25519 surge da necessidade de melhoria de performance e segurança dos algoritmos de *Elliptic Curve Cryptography* (ECC) [45].

  Este algoritmo permite efetuar a troca de de uma chave secreta através da utilização do método de Diffie-Hellman para a partilha do segredo.

  Assim, este esquema criptográfico ganha um maior nível de segurança para a distribuição de chaves.

5.4.3 *X3DH*

O protocolo Triple Diffie-Hellmen (X3DH) é o protocolo que efetua a troca de chaves de forma a estabelecer uma chave secreta partilhada entre duas entidades [46].

Esta chave secreta partilhada é utilizada para a cifragem no processo de troca de mensagens entre as entidades que conhecem esta.

5.4.4 *Double Ratchet*

A utilização da componente do protocolo Signal, Double Ratchet, permite assegurar alguma características de segurança, como por exemplo:

- Resiliência;
- *Forward security*;
- *Break-in recovery.*

A chave secreta partilhada através da utilização do X3DH é utilizada para os dois *ratchets*, um de envio e um de resposta, propostos pelo protocolo *Signal*.

Esta componente prevê ainda que os *ratchets* de ambas as entidades estejam sincronizados, isto é, quando é enviada uma mensagem por parte de uma entidade, deslocando o *ratchet* de envio, a outra entidade terá de alterar o *ratchet* de receção também.

5.5 IMPLEMENTAÇÃO DO PROTOCOLO

Nesta secção prevê-se a explicação do «protocolo» proposto elaborado no âmbito do projeto, que visa a utilização de componentes do protocolo *Signal*, com algumas alterações face às condições de implementação.

A aplicação de *chat* foi desenvolvida com o objetivo de ter duas entidades na mesma rede a comunicar através do protocolo elaborado neste projeto, tendo por base o protocolo *Signal* e os seus componentes, sendo que uma limitação é a indisponibilização da comunicação de duas entidades em redes diferentes.

5.5.1 *Criação entidade*

Para o correto funcionamento da aplicação de *chat*, é necessária a criação de entidades, isto é, nós que irão estar ligados numa rede Peer-To-Peer (P2P), elementos chave na aplicação.

Uma entidade é composta por várias chaves bem como por conjuntos (*bundles*) de chaves que são utilizados para o reconhecimento dos outros nós na rede. Permite ainda guardar as chaves utilizadas para a comunicação.

A Listagem 1 apresenta os atributos de uma entidade:

Listagem 1: Atributos da entidade

```
name = undefined;
hasKeys = false;
// Ed25529Keys
identityKeys = undefined;
// x25519Keys
preKeys = undefined;
// x25519 Signature
preKeySignature = undefined;
//one time keys
oneTimeKeysCount = 0
oneTimeKeysExtractedCount = 0
oneTimeKeys = undefined
extractedOneTimeKeys = undefined

// Relationships
existingBundles = undefined
sendingBundles = undefined
receivingBundles = undefined
```

É possível verificar os diversos atributos que caracterizam uma entidade na aplicação de *chat*, desde as chaves utilizadas para o processo de cifragem das mensagens com recurso ao protocolo X3DH e ao Double Ratchet, através dos atributos *sendingBundles* e *receivingBundles*.

O atributo *existingBundles* mantém os *bundles* das entidades que receberam pedido de criação de *bundle*, pois este é chave para a cifragem dos dados, seguindo o protocolo proposto.

- Geração de chaves:

  Para efetuar a comunicação entre as várias entidades na rede de forma segura, são necessárias chaves que garantem a cifragem da informação e por consequência uma maior segurança da informação enviada entre os vários utilizadores.

As chaves utilizadas na aplicação de *chat* são dos tipos ED25519 e X25519. Adicionalmente, também são geradas chaves One Time Password de forma a obter maior segurança. Estas chaves OTP são utilizadas na criação da entidade e para validação do *bundle* partilhado com a outra entidade.

- Criação *sockets*:

  A aplicação de *chat* foi desenvolvida com o objetivo de ser Pure Peer-To-Peer de forma a retirar entidades intermédias na comunicação da mensagem ou mesmo nas trocas de chaves entre as entidades da rede.

  Para este efeito, a aplicação foi desenvolvida através da utilização da *framework* socket.io de forma a criar *sockets* que se interligam diretamente, sendo esta a base da ligação entre os vários nós da rede.

- Criação *bundle*:

  O *bundle* é uma peça fundamental na aplicação de *chat*. É utilizado para a troca das chaves públicas entre as diversas entidades.

  O *bundle* é composto por um nome de referencia do *bundle* (*identity*), uma chave do tipo ED25529 (identityKey), uma chave do tipo X25519 (preKey), uma assinatura da chave do tipo X25529 (preKeySignature) e uma chave do tipo OTP.

  Estas chaves são utilizadas nos métodos de decifragem das mensagens recebidas de uma determinada entidade. Esta entidade é a responsável pela criação e disponibilização do *bundle* para as outras entidades que efetuem o pedido de criação de *bundle*.

A Figura 22 apresenta o esquema de arranque da aplicação de *chat* e exemplifica os conceitos anteriormente apresentados. É identificada a etapa de criação das chaves pelas diversas entidades, bem como o fluxo das mensagens, sendo que existe diferenças no processo de envio e receção da primeira mensagem para as mensagens subsequentes.

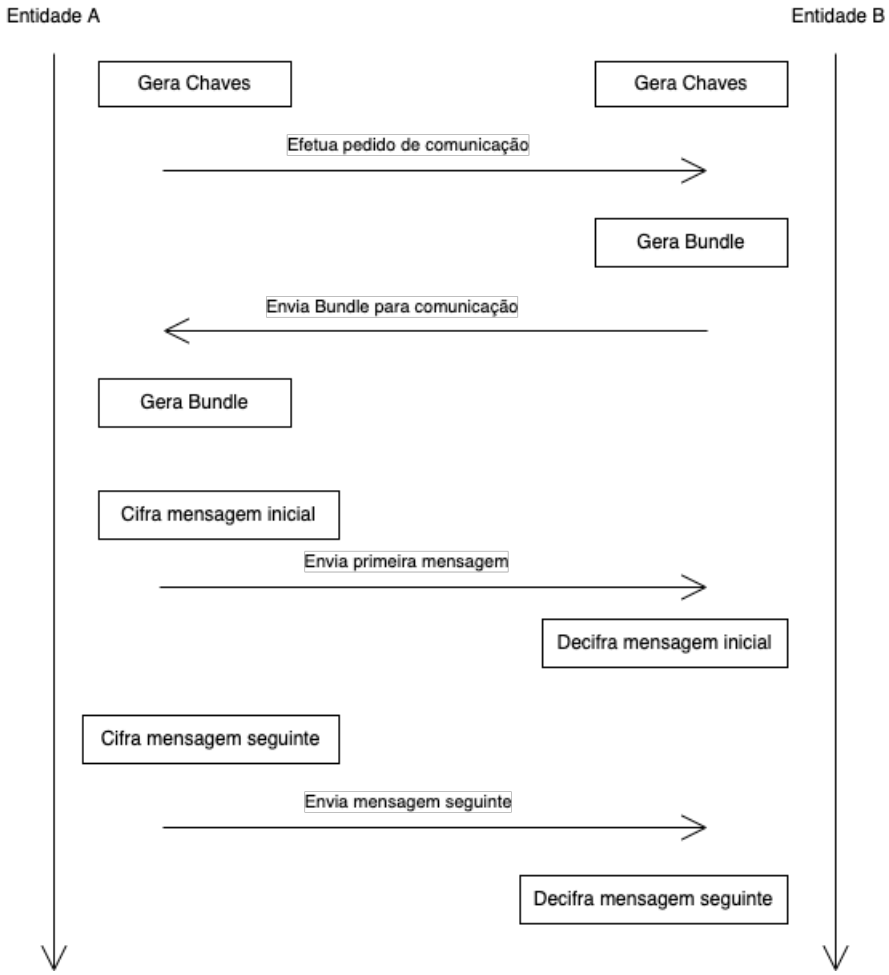

Figura 21: Esquema do funcionamento base da aplicação

5.5.2 *Gestão de mensagens*

Devido à necessidade de serem criadas chaves para todas as mensagens enviadas, foram criados métodos diferentes para o envio e receção das mensagens. Assim, existem métodos para envio e receção da mensagem inicial («createInitialMessageAsync» e «processInitialMessageAsync») e métodos para envio e receção das mensagens subsequentes («createNextMessageAsync» e « processNextMessageAsync»).

5.5.2.1 *Envio mensagem inicial*

Para o envio e receção da primeira mensagem, é enviada a mensagem do Renderer (página web) para o Processor (servidor local) através do método Inter Processor Comunication (IPC). Assim que o servidor recebe a mensagem, aplica o método

de cifragem para a primeira mensagem. Este método varia entre a primeira e a segunda iteração pois na primeira iteração é necessário instanciar as chaves e os *ratchets* do Double Ratchet. Após a cifragem da mensagem, esta é enviada para o nó correspondente.

A Listagem 2 representa o código para o envio da primeira mensagem. É possível verificar a utilização do *bundle* de chaves da entidade que se pretende enviar a mensagem da linha 9 à linha 14. De seguida aplica-se o X3DH, com a geração das 4 variáveis «DHx» nas linhas 17 a 20. Após o X3DH cria-se a chave secreta. Por fim, cifra-se a menagem com a chave secreta gerada anteriormente e atualiza-se a mesma de forma a ficar preparada para a próxima mensagem.

Listagem 2: Cifragem mensagem inicial

```
static async createInitialMessageAsync(recipientBundle, senderEntity, message) {
    const sodium = new SodiumWrapper();

    const valid = await sodium.verifyBundleAsync(recipientBundle);
    if (!valid) {
        throw new Error("Invalid Signature")
    }

    const IKa = await
    ↪  sodium.x25519PrivateFromEd25519(senderEntity.identityKeys.privateKey)
    const SPKb = await sodium.x25519PublicFromEd25519(await
    ↪  sodium.encodeToBinAsync(recipientBundle.preKey))
    const ephemeral = await sodium.generatex25519KeysAsync();
    const EKa = ephemeral.privateKey
    const IKb = await sodium.x25519PublicFromEd25519(await
    ↪  sodium.encodeToBinAsync(recipientBundle.identityKey))
    const OPKb = await sodium.x25519PublicFromEd25519(await
    ↪  sodium.encodeToBinAsync(recipientBundle.oneTimeKey))


    const DH1 = await sodium.ScalarMultAsync(IKa, SPKb)
    const DH2 = await sodium.ScalarMultAsync(EKa, IKb)
    const DH3 = await sodium.ScalarMultAsync(EKa, SPKb)
    const DH4 = await sodium.ScalarMultAsync(EKa, OPKb)

    const SK = await sodium.getInitialSecretKey(DH1, DH2, DH3, DH4)

    const associatedData = await sodium.encodeToHexAsync(
        Buffer.concat([
            senderEntity.identityKeys.publicKey.getBuffer(),
            IKb.getBuffer()
        ]))


    const { encryptedMessage, newSK } = await sodium.encryptAsync(message, SK,
    ↪  associatedData)
    senderEntity.storeEntitySending(recipientBundle, associatedData, newSK)

    return {
        "sender": senderEntity.name,
        "senderPublicKey": await sodium.encodeToHexAsync(senderEntity.identityKe
        ↪  ys.publicKey.getBuffer()),
        "ephemeralKey": await
        ↪  sodium.encodeToHexAsync(ephemeral.publicKey.getBuffer()),
        "encryptedMessage": encryptedMessage
    }

}
```

5.5.2.2 *Receção mensagem inicial*

O servidor do nó correpondente, ao receber a mensagem cifrada, aplica o método para a decifragem da primeira mensagem recebida. À semelhança do método de envio da primeira mensagem, este também requer que seja instanciadas as chaves e os *ratchets*. O processo de decifragem requer a criação de chaves intermédias utilizando o protocolo X3DH, de forma a calcular o segredo conhecido SK. Como as chaves são simétricas , a chave SK vai ser a mesma que foi utilizada para cifrar a mensagem

enviada. Deste modo é possível decifrar corretamente a mensagem enviada pela outra entidade. Assim que termina o processo de decifragem, o servidor (Processor) envia para a página *web* (Renderer) através do IPC a mensagem de modo a que esta seja atualizada na página e fique visível para o cliente.

A Listagem 3 representa o código para a receção da primeira mensagem. Este processo é bastante semelhante ao anterior sendo que se verifica a maior diferença na linha 25, onde é aplicada a decifragem da mensagem cifrada recebida e atualizada a chave secreta, gerada antes da aplicação deste método, de modo a estar preparada para a próxima mensagem recebida.

Listagem 3: Decifragem mensagem inicial

```
static async processInitialMessageAsync(receivedMessage, recipientEntity) {
    const sodium = new SodiumWrapper();

    const SPKb = await
    ↪  sodium.x25519PrivateFromEd25519(recipientEntity.preKeys.privateKey)
    const IKa = await sodium.x25519PublicFromEd25519(await
    ↪  sodium.encodeToBinAsync(receivedMessage.senderPublicKey))
    const IKb = await
    ↪  sodium.x25519PrivateFromEd25519(recipientEntity.identityKeys.privateKey)
    const EKa = await sodium.x25519PublicFromBuffer(await
    ↪  sodium.encodeToBinAsync(receivedMessage.ephemeralKey))

    const oneTimeKeyForSender = recipientEntity.extractedOneTimeKeys.find(k =>
    ↪  k.entity == receivedMessage.sender)
    const OPKb = await
    ↪  sodium.x25519PrivateFromEd25519(oneTimeKeyForSender.key.privateKey)

    const DH1 = await sodium.ScalarMultAsync(SPKb, IKa)
    const DH2 = await sodium.ScalarMultAsync(IKb, EKa)
    const DH3 = await sodium.ScalarMultAsync(SPKb, EKa)
    const DH4 = await sodium.ScalarMultAsync(OPKb, EKa)

    const SK = await sodium.getInitialSecretKey(DH1, DH2, DH3, DH4)

    const associatedData = await sodium.encodeToHexAsync(
        Buffer.concat([await
        ↪  sodium.encodeToBinAsync(receivedMessage.senderPublicKey), (await
        ↪  sodium.x25519PublicFromEd25519(recipientEntity.identityKeys.publicKe
        ↪  y)).getBuffer()])
    );

    try {

        const { decryptedMessage, newSK } = await
        ↪  sodium.decryptAsync(receivedMessage.encryptedMessage, SK,
        ↪  associatedData)

        recipientEntity.storeEntityReceiving(receivedMessage.sender,
        ↪  associatedData, newSK)

        return decryptedMessage.toString();

    } catch (error) {
        console.log("ERROR Decrypting Data")
        throw error
    }

}
```

5.5.2.3 *Envio de mensagens seguintes*

Para as restantes mensagens, o processo de envio e receção é semelhante sendo a única diferença os métodos de cifragem e decifragem das mensagens, tal como se pode ver na Listagem 4.

A Listagem 4 representa o código para o envio das mensagens seguintes. Para as mensagens seguintes, o processo apenas necessita do *bundle* de chaves gerado no envio da mensagem anterior. Assim, utiliza se este conjunto de chaves para aplicar

a cifragem da mensagem (linha 8) e atualiza-se a chave secreta para a próxima utilização.

Listagem 4: Cifragem mensagens seguintes

```
1  static async createNextMessageAsync(recipientBundle, senderEntity, message) {
2      const sodium = new SodiumWrapper();
3
4      const sendingBundle = senderEntity.sendingBundles.find(b => b.identity ==
       ↪  recipientBundle.identity)
5      const SK = sendingBundle.SK
6      const associatedData = sendingBundle.associatedData
7
8      const { encryptedMessage, newSK } = await sodium.encryptAsync(message, SK,
       ↪  associatedData)
9      senderEntity.updateSKSending(recipientBundle.identity, newSK)
10
11     return {
12         "sender": senderEntity.name,
13         "encryptedMessage": encryptedMessage
14     }
15
16 }
```

5.5.2.4 *Receção das mensagens seguintes*

Neste ponto, como já foram instanciadas as chaves e os *ratchets* quer de envio, quer de receção, simplesmente é gerada uma nova chave com base na anterior, fazendo com que seja atualizado o *ratchet* de envio, para o nó que envia a mensagem, e o *ratchet* de receção, para o nó que recebe a mensagem.

A Listagem 5 representa o código para a receção das mensagens seguintes. Tal como para a primeira mensagem recebida, este método não difere muito do processo de cifragem das mensagens seguintes. No entanto, verifica-se uma diferença no método invocado na linha 10 que aplica a decifragem da mensagem inicial e refresca a chave secreta para o próximo uso.

Listagem 5: Decifragem mensagens seguintes

```
static async processNextMessageAsync(receivedMessage, recipientEntity) {
    const sodium = new SodiumWrapper();

    const receivingBundle = recipientEntity.receivingBundles.find(b =>
    ↪  b.identity == receivedMessage.sender)
    const SK = receivingBundle.SK
    const associatedData = receivingBundle.associatedData

    try {

        const { decryptedMessage, newSK } = await
        ↪  sodium.decryptAsync(receivedMessage.encryptedMessage, SK,
        ↪  associatedData)

        recipientEntity.updateSKReceiving(receivedMessage.sender, newSK)

        return decryptedMessage.toString();

    } catch (error) {
        console.log("ERROR Decrypting Data")
        throw error
    }

}
```

A Figura 22 representa esquematicamente o processo de envio da primeira mensagem.

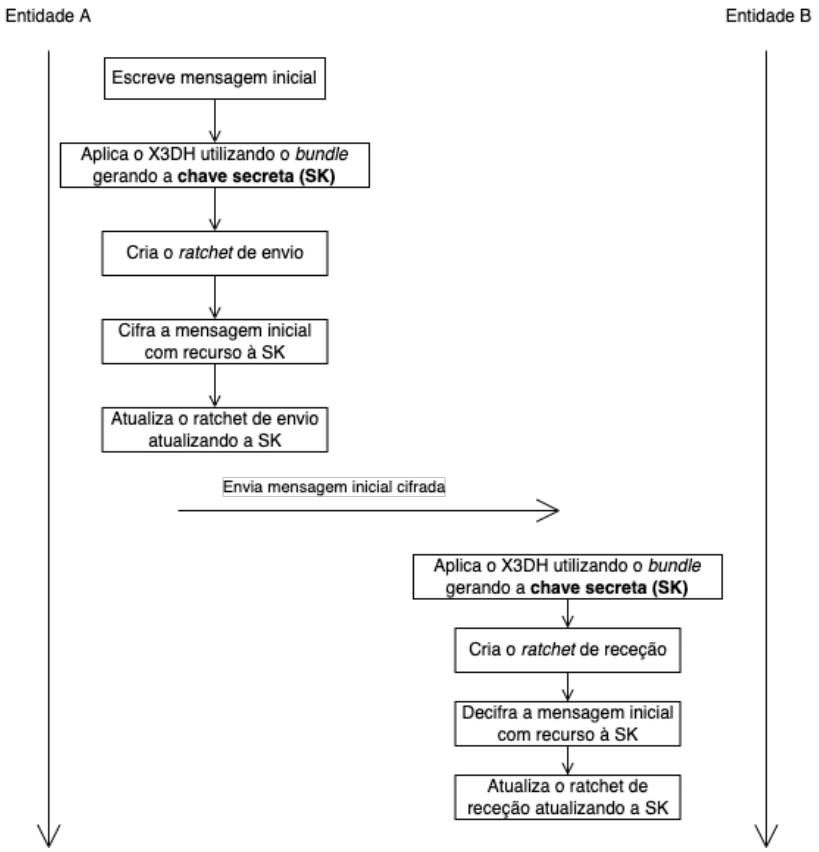

Figura 22: Envio da mensagem inicial

Este começa com a aplicação do protocolo X3DH, utilizando as seguintes chaves:

- Chave privada da entidade que envia a mensagem (IKa);
- Chave efémera (uma utilização apenas) (EKa);
- Chave pública da outra entidade do tipo ED25519 (IKb);
- Chave publica da outra entidade do tipo X25519 (SPKb);
- Chave do tipo OTP da outra entidade (OPKb).

De seguida, após a geração das chaves Diffie-Hellman, é aplicada uma Key Derivation Function (KDF) sobre a combinação das quatro chaves geradas no processo anterior. O resultado da aplicação da KDF é o segredo partilhado (SK) entre as duas entidades. Está SK é utilizada para efetuar a cifragem da mensagem e desta forma assegurar segurança da informação enviada entre as diversas entidades.

Por fim, antes do envio da mensagem a SK é atualizada, sendo representada pelo movimento do *ratchet*.

A Figura 23 apresenta o fluxo do envio das mensagens seguintes.

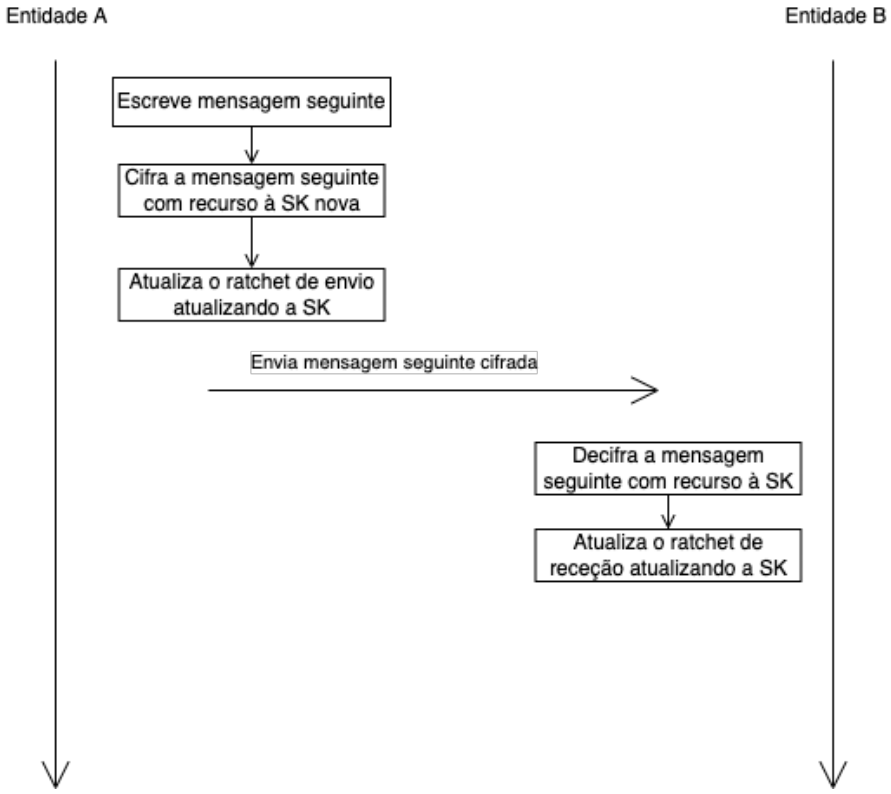

Figura 23: Envio da mensagem seguinte

Para o fluxo das mensagens seguintes o processo é um pouco mais simplificado uma vez que não exige a criação dos *ratchets* pois já tinham sido criados na etapa inicial. Assim, para este fluxo apenas é necessária a utilização da chave secreta SK gerada anterior para cifrar a mensagem seguinte, sendo que, no final de cada processo de cifragem, a chave secreta SK é novamente atualizada, de modo a garantir nova chave para o próximo processo de cifragem de dados.

5.5.3 *Armazenamento chaves*

Nesta subsecção pretende-se abordar os temas relacionados com o armazenamento das chaves das entidades criadas localmente pela aplicação, quer as recebidas via *bundle* por outras entidades.

Uma vez que as entidades trocam *bundles* de chaves entre si, estas são guardadas aplicacionalmente na entidade. Assim verifica-se que as entidades armazenam as chaves consigo.

No arranque, durante as alterações, como por exemplo na geração/receção de um novo *bundle* ou nos processos criptográficos que face ao Double Ratchet, atualiza

a chave através de Key Derivation Functions gera nova chave, e no término da aplicação é armazenada a estrutura da entidade em ficheiro de texto.

Este ficheiro é lido no arranque da aplicação de forma a preencher novamente a estrutura da entidade e desta forma evita que sejam perdidas as chaves já partilhadas e as iterações dos *ratchets*. Caso não exista ficheiro a entidade é criada novamente, perdendo todas as chaves partilhadas com outras entidades.

5.6 SÍNTESE

No seguimento da proposta deste projeto, protocolo seguro de mensagens assíncronas, foi possível demonstrar que é possível a utilização do protocolo Signal sem que exista a necessidade de uma terceira entidade, isto é, permitir a comunicação utilizando uma arquitetura Pure Peer-to-Peer (PP2P).

Para validar estes objetivos são necessárias condições especiais, sendo uma das mais importantes, a ligação entre as entidades ter de ocorrer dentro da mesma rede, pelo que, de forma a aumentar a segurança, recomenda-se a utilização da mesma rede para a comunicação das entidades que utilizam a aplicação.

Ao longo do desenvolvimento foram detetadas algumas melhorias ao funcionamento da aplicação de *chat* que visam alterar a performance, as funcionalidades e segurança da mesma. Uma destas melhorias que visa o aumento da segurança da aplicação de *chat* PP2P é a alteração do servidor criado do lado do cliente. Este é criado ficando à escuta de pedidos Hypertext Transfer Protocol (HTTP), no entanto, este protocolo não cifra os pacotes enviados pela rede, pelo que se sugere que futuramente seja utilizado o protocolo Hypertext Transfer Protocol Secure (HTTPS), também conhecido por *HTTP over TLS*, para a ligação entre os clientes, representados por outras entidades na rede, e o servidor. Este protocolo aplica o protocolo HTTP em conjunto com os protocolos Secure Sockets Layer (SSL) e Transport Layer Security (TLS) que por sua vez aplicam a segurança dos dados enviados, encapsulando estes de forma a evitar ataques de MITM [47].

Outro aspeto que pode ser melhorado é o funcionamento dos pedidos de conexão. Atualmente a aplicação de *chat* requer que seja introduzido o endereço da entidade que se pretende comunicar de forma a obter e gerar um *bundle*. Assim, sugere-se a implementação de um sistema que permita identificar outras entidades presentes na rede e permitir o utilizador selecionar a entidade com a qual pretende comunicar. Esta alteração requer a modificação do processo de instanciação da aplicação de *chat*

pois esta cria as entidades no arranque da mesma. Este ponto é indicado como aspeto a melhorar na aplicação, no entanto, não é necessário para provar o funcionamento do «protocolo» implementado.

Verificou-se que a aplicação desenvolvida não aplica uma das funcionalidades do protocolo *Signal*, o envio assíncrono das mensagens entre os utilizadores. Este ponto requer que seja efetuada uma análise de como efetuar a distribuição de mensagens quando o utilizador destino não está presente para receber a mensagem. Assim, sugere-se que seja implementada uma listagem dos utilizadores presentes na rede de forma a distribuir a mensagem por um dos nós presentes na rede. No entanto, esta medida pode funcionar para este caso em concreto uma vez que o número de utilizadores é reduzido, no entanto, pode ser prejudicial quando o número de nós na rede é maior pois acarreta maiores encargos para os nós que estiverem responsáveis por entregar as mensagens pendentes ao destino.

Outra sugestão para este ponto é a utilização das bases de dados distribuídas também já mencionadas para modificar o método de armazenamento das chaves privadas e públicas das entidades. Estas podem armazenar as mensagens cifradas enviadas entre os utilizadores nos respetivos dispositivos garantido assim o armazenamento do histórico de mensagens trocadas entre os utilizadores e habilitando a característica de assincronismo nas mensagens enviadas.

Futuramente, de forma a melhorar esta situação, sugere-se que as chaves sejam armazenadas numa base de dados distribuída, sendo que cada entidade tem a sua instância local. Esta alteração permite retirar a existência do ficheiro TXT da aplicação, no entanto, esta fica dependente da base de dados local para a obtenção das chaves necessárias para a comunicação.

A utilização de bases de dados distribuídas permite ainda resolver outro tema relacionado com as mensagens assíncronas. Atualmente, caso um utilizador se desconecte da conversa com outra entidade e sejam enviadas uma ou mais mensagens para esse utilizador durante o período em que este esteve *offline*, vai existir um assincronismo no *ratchet* de envio da entidade que envia as mensagens. Propomos solucionar este tema com a utilização das bases de dados descentralizadas pois para alem de se enviar os bundles para o servidor da base de dados, tambem se pode enviar as mensagens cifradas, ordenadas para que a decifragem seja efetuada corretamente, com a indicação de qual a entidade destino e assim, quando a entidade voltar a conectar-se à aplicação de *chat*, esta obter os dados para atualizar as mensagens do lado da entidade destino e o *ratchet* de receção.

Por fim, é possível afirmar que se consegue estabelecer conexão entre duas entidades que estejam na mesma rede utilizando um protocolo bastante semelhante ao protocolo Signal, em que a maior diferença é a desativação do servidor «central» que contém as chaves dos utilizadores da aplicação, em que previamente existe a necessidade da ligação a este antes da mensagem ser enviada para a entidade pretendida.

6

CONCLUSÕES

Através da realização deste projeto foi possível colocar em prática alguns dos conhecimentos adquiridos ao longo do Mestrado e ainda permitiu a aprendizagem do funcionamento e aplicação de diversas tecnologias como ElectronJS, *Signal* e JavaScript. Permitiu aprender conceitos e características especificas do protocolo *Signal*, quer ao nível de segurança, quer ao nível da sua implementação. Também providenciou conhecimento ao nível das diferentes arquiteturas P2P existentes, apesar de se ter optado pela Pure Peer-To-Peer. Por fim permitiu explorar e aprofundar o conhecimento ao nível das aplicações de MI uma vez que existiu a necessidade de verificar as várias características de aplicações desta natureza de forma a efetuar uma escolha mais informada e contextualizada.

Adicionalmente, verificou-se ser possível a criação de uma aplicação de *chat* PP2P implementando um «protocolo» semelhante ao protocolo *Signal*, mantendo as principais características de segurança deste, mas sem a necessidade de um servidor que faz de ponto intermédio entre as mensagens enviadas pelos utilizadores e de armazenamento das chaves públicas das diversas entidades.

Apesar de se ter evidenciado a aplicabilidade do protocolo proposto, que se traduziu na prova de conceito, consegue-se identificar alguns aspetos de melhoria à aplicação, pelo que se verificam nos seguintes pontos:

- **Alteração do processo de inicio de comunicação**:

  Sugere-se a alteração do funcionamento da aplicação de forma que a automaticamente seja possível encontrar os nós que se encontrem na rede local do utilizador, reduzindo assim a manualidade da aplicação.

- **Alteração do método de armazenamento das chaves utilizadas nos processos criptográficos**:

  Pretende-se que seja melhorado o processo de armazenamento das chaves privadas e públicas das entidades. Atualmente estas são utilizadas nos processos criptográficos aplicados às trocas de chaves efetuadas pela aplicação e às mensagens enviadas e recebidas pelos utilizadores da mesma.

- **Suporte para mensagens assíncronas**:

Atualmente a aplicação desenvolvida apenas permite a troca de mensagens entre os utilizadores de forma síncrona, isto é, ambos os utilizadores tem de estar ativos na rede para conseguirem trocar mensagens entre si. Desta forma, caso seja suportado o envio de mensagens assíncronas, deixa de existir a necessidade de ambos estarem em simultâneo na rede local, disponibilizando as mensagens enviadas sem ligação entre os nós quando efetuado próximo inicio de sessão.

- **Inclusão do envio de anexos nas mensagens**:

  Esta funcionalidade permitirá às entidades participantes enviarem anexos entre si, sendo que será necessário validar as alterações ao nível do «protocolo» implementado.

- **Implementação de chamadas de voz**:

  O suporte desta funcionalidade iria habilitar os utilizadores a efetuarem chamadas de voz entre si. Para a implementação desta funcionalidade terá de ser analisado o protocolo a utilizar de forma a ser possível a integração com o «protocolo» desenvolvido para a aplicação de *chat* PP2P.

- **Adição da funcionalidade de grupos**:

  A adição dos grupos no chat permitirá aos utilizadores criarem grupos para troca de mensagens e, em conjunto com o ponto anterior, efetuar chamadas de grupo. Para estes funcionalidades terá de ser testado o «protocolo» implementado e a forma de integração com o protocolo escolhido para efetuar as chamadas de voz, tal como mencionado no ponto anterior.

Tal como mencionado anteriormente, foi extremamente proveitoso para a aprendizagem generalizada das componentes envolvidas na pesquisa da informação necessária para o desenvolvimento da aplicação de *chat* e implementação do protocolo para evidenciar os conceitos propostos.

Por fim, e embora existam vários pontos a melhorar, considera-se que uma abordagem semelhante a que foi proposta neste projeto permitiria melhorar substancialmente a segurança e privacidade dos utilizadores de aplicações de Mensagens Instantâneas. Assim sugere-se que exista mais investigação nesta área pois merece que existam processo que consigam aplicar cada vez mais segurança nos seus processos de forma transparente para o utilizador.

BIBLIOGRAFIA

[1] Moxie Marlinspike, Trevor Perrin, *Signal specifications*, https://signal.org/docs/, Online; acessado 23 fevereiro 2022, 2021.

[2] A. de Mauro, «A peer-to-Peer network for real-time multiple-description video communications using multiple paths», 2006.

[3] Cisco Systems, Inc, *What Is Cybersecurity*, https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html, Online; acessado 4 março 2022, 2022.

[4] ENISA, «ENISA Threat Landscape 2021 — ENISA», 2021. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2021.

[5] Simon Kemp, *Digital 2021: Global Overview Report*, https://datareportal.com/reports/digital-2021-global-overview-report, Online; acessado 3 março 2022, 2021.

[6] Mark Williams, *Secure Messaging Apps Comparison*, https://www.securemessagingapps.com/, Online; acessado 8 março 2022, 2021.

[7] Apple Inc., *APNs Overview*, https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html, Online; acessado 28 março 2022, 2018.

[8] Cloudflare, Inc., *What is TLS (Transport Layer Security)?*, https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/, Online; acessado 28 março 2022, 2022.

[9] S. Coull e K. Dyer, «Privacy Failures in Encrypted Messaging Services: Apple iMessage and Beyond», mar. de 2014. URL: http://arxiv.org/abs/1403.1906.

[10] Element, *Element Matrix Services*, https://element.io/matrix-services, Online; acessado 17 março 2022, 2022.

[11] ——, *Element*, https://element.io/open-source, Online; acessado 17 março 2022, 2022.

[12] Moxie Marlinspike, Trevor Perrin, *The X3DH Key Agreement Protocol*, https://signal.org/docs/specifications/x3dh/x3dh.pdf, Online; acessado 19 Junho 2020, 2016.

[13] ——, *The Double Ratchet Algorithm*, https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf, Online; acessado 19 Junho 2020, 2016.

[14] Microsoft, *Skype*, https://www.skype.com/en/features/, Online; acessado 17 março 2022, 2022.

[15] Telegram team, *What is Telegram? What do I do here?*, https://telegram.org/faq, Online; acessado 28 março 2022, 2022.

[16] ——, *MTProto Mobile Protocol*, https://core.telegram.org/mtproto, Online; acessado 28 março 2022, 2022.

[17] Threema GmbH., *Threema Is a Statement*, https://threema.ch/en/about, Online; acessado 28 março 2022, 2022.

[18] Viber Media S.à r.l., *About Viber*, https://www.viber.com/en/about/, Online; acessado 28 março 2022, 2022.

[19] Wickr.Inc, *About Us*, https://wickr.com/about-us/, Online; acessado 28 março 2022, 2022.

[20] Wire Swiss GmbH, *About*, https://wire.com/en/about/, Online; acessado 28 março 2022, 2022.

[21] Oxen, *Session*, https://docs.oxen.io/products-built-on-oxen/session, Online; acessado 28 março 2022, 2021.

[22] Assembleia da República, *Lei n.º 58/2019*, https://dre.pt/dre/detalhe/lei/58-2019-123815982, Online; acessado 27 março 2022, 2019.

[23] J. B. F. C. Marques e J. Jorge, «Authority 2.0», 2020.

[24] H. H. Kseniia Ermoshina Francesca Musiani, «End-to-End Encrypted Messaging Protocols: An Overview», *Third International Conference, INSCI 2016 - Internet Science*, 2016, https://hal.inria.fr/hal-01426845/document.

[25] M. Pound. (2019). Instant Messaging and the Signal Protocol - Computerphile, Computerphile, URL: https://www.youtube.com/watch?v=DXv1boalsDI.

[26] ——, (2019). Double Ratchet Messaging Encryption - Computerphile, Computerphile, URL: https://www.youtube.com/watch?v=9sO2qdTci-s.

[27] J. S. Paul Rosler Christian Mainka, «More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema», *2018 IEEE European Symposium on Security and Privacy*, 2018, https://ieeexplore.ieee.org/document/8406614.

[28] NIST, *NIST Confidentiality*, https://csrc.nist.gov/glossary/term/confidentiality, Online; acessado 26 fevereiro 2022.

[29] ——, *NIST Integrity*, https://csrc.nist.gov/glossary/term/integrity, Online; acessado 26 fevereiro 2022.

[30] ——, *NIST Availability*, https://csrc.nist.gov/glossary/term/availability, Online; acessado 26 fevereiro 2022.

[31] ——, *NIST Resilience*, https://csrc.nist.gov/glossary/term/resilience, Online; acessado 26 fevereiro 2022.

[32] Scrum.org, *What is Scrum?*, https://www.scrum.org/resources/what-is-scrum, Online; acessado 27 março 2022, 2022.

[33] Signal Messenger, LLC., *Libsignal*, https://github.com/signalapp/libsignal, Online; acessado 27 março 2022, 2022.

[34] Soatok, *Going Bark: A Furry's Guide to End-to-End Encryption*, https://soatok.blog/2020/11/14/going-bark-a-furrys-guide-to-end-to-end-encryption/, Online; acessado 27 março 2022, 2020.

[35] Paragon Initiative Enterprises, *Sodium-Plus*, https://github.com/paragonie/sodium-plus, Online; acessado 27 março 2022, 2021.

[36] Socket.IO, *Socket.io*, https://socket.io/, Online; acessado 27 março 2022, 2022.

[37] Caleb, *AlpineJS*, https://alpinejs.dev/, Online; acessado 27 março 2022, 2022.

[38] Bootstrap Team, *Build fast, responsive sites with Bootstrap*, https://getbootstrap.com/, Online; acessado 27 março 2022, 2022.

[39] OpenJS Foundation, Electron contributors, *Electron*, https://www.electronjs.org/docs/latest/, Online; acessado 23 fevereiro 2022, 2021.

[40] Google, *Chromium*, https://www.chromium.org/chromium-projects/, Online; acessado 23 fevereiro 2022.

[41] OpenJS Foundation, *Node.js*, https://nodejs.org/en/, Online; acessado 23 fevereiro 2022.

[42] OpenJS Foundation, Electron contributors, *Electron process model*, https://www.electronjs.org/docs/latest/tutorial/process-model, Online; acessado 23 fevereiro 2022, 2021.

[43] libsodium team, *Libsodium*, https://doc.libsodium.org/, Online; acessado 27 março 2022, 2021.

[44] S. Josefsson e I. Liusvaara, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, RFC 8032, jan. de 2017. DOI: 10.17487/RFC8032. URL: https://www.rfc-editor.org/info/rfc8032.

[45] A. Langley, M. Hamburg e S. Turner, *Elliptic Curves for Security*, RFC 7748, jan. de 2016. DOI: 10.17487/RFC7748. URL: https://www.rfc-editor.org/info/rfc7748.

[46] F. van der Have, «The X3DH Protocol: A Proof of Security», tese de doutoramento, RADBOUD UNIVERSITY, 2022.

[47] Martin Thoma, *HTTPS vs SSL vs TLS*, https://medium.com/plain-and-simple/https-vs-ssl-vs-tls-8a0ad0604276, Online; acessado 18 março 2022, 2021.

APÊNDICES

APÊNCICE A

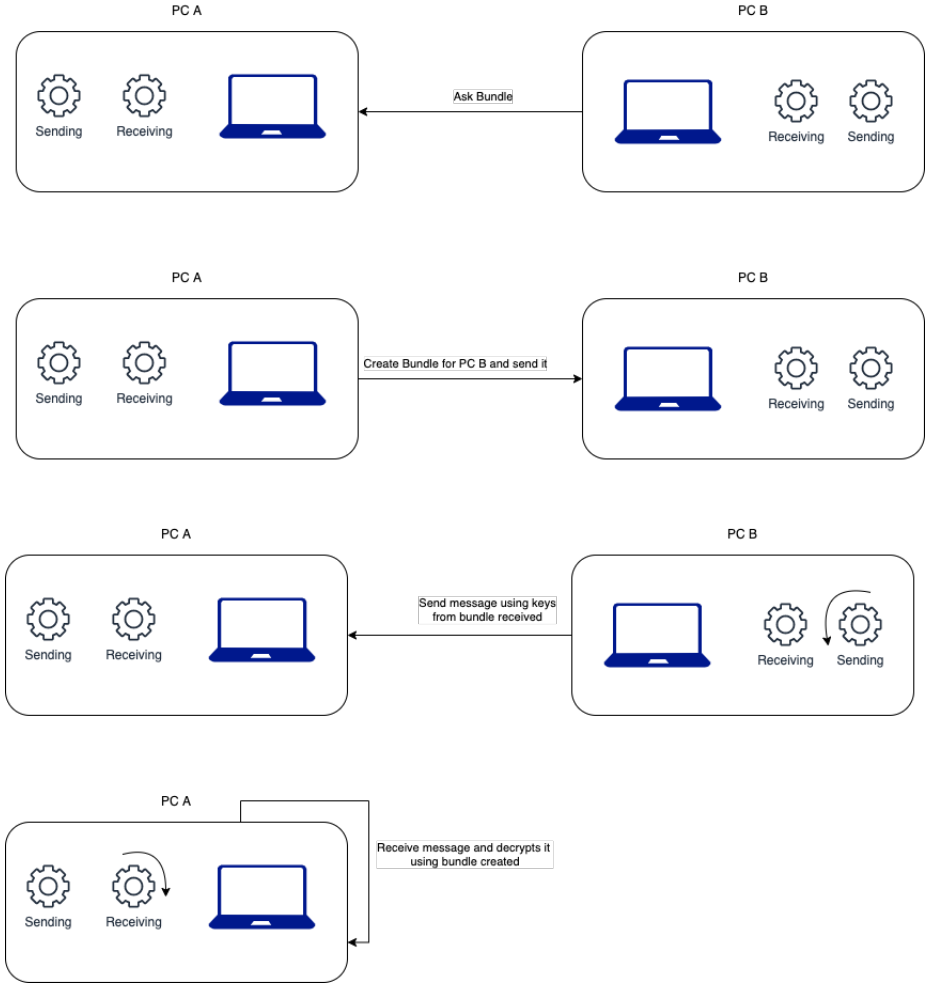

Figura 24: Arquitetura da comunicação

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Protocolo seguro de mensagens instantâneas"*, é original e foi realizado por João Bernardo Gomes Jorge (2190373) sob orientação de Professora Doutora Marisa Maximiano (marisa.maximiano@ipleiria.pt) e de Professor Ricardo Gomes (ricardo.p.gomes@ipleiria.pt).

Leiria, março de 2022

Assinado por: João Bernardo Gomes Jorge
Identificação: BI15144803
Data: 2022-03-30 às 23:48:50

João Bernardo Gomes Jorge