

# Análise da Criptografia Ponta-a-Ponta: Um Estudo sobre o Protocolo Signal e suas implementações

Luiz Antônio Lima de Freitas Leite<sup>1</sup>, Max José Lobato Pantoja Junior<sup>1</sup>,  
Wesley Pontes Barbosa<sup>1</sup>

<sup>1</sup>Instituto de Ciências Exatas e Naturais (ICEN) – Universidade Federal Pará  
Belém, PA – Brasil

{luiz.freitas.leite,max.junior,wesley.pontes.barbosa}@icen.ufpa.br

**Abstract.** *[TODOS]/[REVISAR para garantir consistencia e coesao]* This paper analyzes the Signal Protocol, the *de facto* standard for secure messaging. We explore its evolution from the "Axolotl"ratchet, named after the self-healing salamander, to its current state. Technical concepts such as the Double Ratchet Algorithm and X3DH are detailed. We present case studies on WhatsApp, discussing its closed-source implementation allowed by specific commercial agreements and the controversy surrounding its 2021 privacy policy update regarding metadata collection. Finally, we examine the Matrix protocol and its Olm/Megolm libraries, which adapt Signal's concepts for federated environments and high-performance group chats.

**Resumo.** *[TODOS]/[REVISAR para garantir consistencia e coesao]* Este artigo analisa o Protocolo Signal, o padrão de fato para mensagens seguras. Exploramos sua evolução desde o algoritmo "Axolotl", nomeado em referência à salamandra regenerativa, até seu estado atual. Conceitos técnicos como o algoritmo Double Ratchet e X3DH são detalhados. Apresentamos estudos de caso sobre o WhatsApp, discutindo sua implementação de código fechado permitida por acordos comerciais específicos e a controvérsia em torno da atualização de sua política de privacidade em 2021 referente à coleta de metadados. Por fim, examinamos o protocolo Matrix e suas bibliotecas Olm/Megolm, que adaptam os conceitos do Signal para ambientes federados e grupos de alta performance.

## 1. Introdução

A comunicação sigilosa é uma necessidade humana tão antiga quanto a própria linguagem, mas o ambiente digital impôs desafios inéditos: uma mensagem enviada por um aplicativo pode trafegar por dezenas de servidores antes de chegar ao destinatário, e cada um desses pontos representa um potencial adversário. Durante anos, a solução predominante foi a criptografia de transporte, representada por protocolos como o TLS/SSL, que protege o canal entre o usuário e o servidor do serviço. Nesse modelo, o conteúdo das mensagens chega descriptografado ao provedor, que tem plena capacidade de lê-las, armazená-las ou entregá-las a terceiros [Antunes and Kowada 2018].

Em 2013, as revelações de Edward Snowden expuseram programas sistemáticos de vigilância conduzidos por agências de inteligência em parceria com grandes empresas de tecnologia, demonstrando que confiar no provedor como guardião das comunicações era uma premissa inadequada. Embora já existissem alternativas como o PGP (*Pretty*

*Good Privacy*), criado em 1991, seu uso exigia conhecimento técnico elevado e uma infraestrutura de gerenciamento de chaves que inviabilizava a adoção pelo público geral.

A resposta a esse impasse veio com a Criptografia Ponta-a-Ponta (E2EE, do inglês *End-to-End Encryption*), paradigma em que as mensagens são cifradas no dispositivo do remetente e decifradas exclusivamente no dispositivo do destinatário, sem que nenhum servidor intermediário tenha acesso ao conteúdo. O protagonista da adoção massiva desse modelo foi o **Protocolo Signal**, cuja solidez matemática foi formalmente verificada por análises independentes [Cohn-Gordon et al. 2020].

### 1.1. História e o Nome Axolotl

O protocolo foi criado por Moxie Marlinspike e Trevor Perrin no âmbito da *Open Whisper Systems*. A tecnologia estreou em 2013 no aplicativo *TextSecure* como alternativa segura ao SMS. Em 2014, a organização firmou um acordo com o WhatsApp para integrar a biblioteca, estendendo a proteção a centenas de milhões de usuários de forma transparente [Marlinspike 2016]. Em 2016, o aplicativo *Signal* reuniu toda a tecnologia sob uma única identidade.

O mecanismo central de criptografia recebeu inicialmente o nome **Axolotl**, em referência ao axolote (*Ambystoma mexicanum*). A analogia é precisa: o axolote é um anfíbio com capacidade documentada de regenerar membros, órgãos e tecido nervoso perdidos. O protocolo possui propriedade equivalente. Se um atacante comprometer as chaves de uma sessão ativa, o algoritmo rotaciona todo o material criptográfico na próxima troca de mensagens, reestabelecendo a segurança da conversa automaticamente, sem qualquer intervenção do usuário [Perrin and Marlinspike 2016]. Em 2016, o nome foi padronizado para **Protocolo Signal**, unificando a nomenclatura da tecnologia e do aplicativo que a popularizou.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta os conceitos criptográficos fundamentais para a compreensão do protocolo; a Seção 3 detalha seu funcionamento interno; a Seção 4 analisa implementações no WhatsApp e no Matrix; e a Seção 5 apresenta as considerações finais.

## 2. Conceitos Básicos

Para compreender o funcionamento do Protocolo Signal, é necessário apresentar os fundamentos criptográficos sobre os quais ele é construído. Esta seção percorre esses conceitos progressivamente, partindo da distinção entre os modelos de criptografia até as propriedades de segurança de alto nível que o protocolo visa garantir.

### 2.1. Criptografia Simétrica e Assimétrica

Na **criptografia simétrica**, a mesma chave é utilizada para cifrar e para decifrar uma mensagem. Matematicamente, dada uma mensagem  $m$  e uma chave secreta  $k$ , a cifra  $c$  é obtida por uma função de encriptação  $E$ , tal que  $c = E_k(m)$ . A recuperação da mensagem original é feita pela função inversa de decriptação  $D$ , onde  $m = D_k(c)$ . O desafio fundamental desse modelo é o *problema da distribuição de chaves*: como dois interlocutores combinam uma chave secreta sem que um adversário que monitore o canal consiga interceptá-la?

A **criptografia assimétrica** (ou de chave pública) resolve esse problema utilizando um par de chaves  $(pk, sk)$ , onde  $pk$  é a chave pública e  $sk$  a privada. Uma mensagem cifrada com a chave pública de um destinatário ( $c = E_{pk}(m)$ ) só pode ser decifrada com sua chave privada correspondente ( $m = D_{sk}(c)$ ). O Protocolo Signal utiliza ambos os modelos em conjunto: a criptografia assimétrica para o estabelecimento seguro de chaves de sessão, e a criptografia simétrica para a cifragem eficiente das mensagens propriamente ditas [Antunes and Kowada 2018].

## 2.2. Diffie-Hellman e Curvas Elípticas (ECDH)

O protocolo Diffie-Hellman (DH), proposto em 1976, permite que duas partes estabeleçam uma chave secreta compartilhada por meio de um canal completamente público, sem qualquer conhecimento prévio uma da outra. A intuição por trás do protocolo pode ser ilustrada por uma analogia com cores: cada parte escolhe uma cor secreta e a combina com uma cor pública comum; o resultado final, obtido por caminhos distintos por cada lado, é idêntico, mas um observador externo vê apenas as cores públicas e não consegue reproduzir a mistura sem conhecer os segredos individuais [Antunes and Kowada 2018].

Formalmente, considere um grupo cíclico  $G$  de ordem prima  $q$  e um gerador  $g$ . Alice escolhe um inteiro aleatório  $a$  (sua chave privada) e calcula  $A = g^a \pmod{p}$  (sua chave pública). Bob escolhe analogamente um inteiro  $b$  e calcula  $B = g^b \pmod{p}$ . O segredo compartilhado  $S$  é então calculado independentemente por ambas as partes:

$$S = B^a = (g^b)^a = g^{ba} = (g^a)^b = A^b \pmod{p} \quad (1)$$

A segurança do DH clássico repousa sobre a dificuldade computacional do *problema do logaritmo discreto*. O Protocolo Signal utiliza uma variante moderna denominada **ECDH** (*Elliptic-Curve Diffie-Hellman*), que opera sobre grupos de pontos em curvas elípticas. Nesse contexto, a operação de exponenciação é substituída pela multiplicação de escalares por pontos na curva, mantendo a mesma lógica de segurança. O ECDH oferece o mesmo nível de segurança do DH tradicional com chaves significativamente menores, o que é essencial para dispositivos móveis com recursos de processamento e bateria limitados.

## 2.3. Funções de Derivação de Chaves (KDF)

Uma **Função de Derivação de Chaves** (KDF, do inglês *Key Derivation Function*) é um componente criptográfico que, a partir de um material de entrada (como uma chave compartilhada obtida via ECDH), produz uma ou mais chaves com as propriedades estatísticas desejadas. O Protocolo Signal emprega especificamente o **HKDF** (*HMAC-based Key Derivation Function*) [Perrin and Marlinspike 2016].

Matematicamente, o HKDF pode ser descrito como uma função que recebe um material de chave de entrada ( $KI$ ), um “sal” opcional e informações de contexto, produzindo uma chave criptograficamente forte ( $KO$ ):

$$KO = \text{HKDF}(salt, KI, info) \quad (2)$$

A propriedade central explorada pelo protocolo é a **unidirecionalidade**: dado o resultado de uma KDF, é computacionalmente inviável reconstituir o material de entrada. Essa característica é o que permite ao Double Ratchet, detalhado na Seção 3, girar as chaves continuamente sem que estados anteriores sejam comprometidos.

## 2.4. Sigilo Perfeito Encaminhado (*Forward Secrecy*)

O **Sigilo Perfeito Encaminhado** (FS, do inglês *Forward Secrecy*) garante que o comprometimento das chaves de longo prazo de um usuário não compromete as mensagens trocadas no passado [Cohn-Gordon et al. 2020]. Para isso, o protocolo gera chaves de sessão *efêmeras*, criadas para uso único e descartadas imediatamente após o uso.

Um exemplo concreto ilustra a importância dessa propriedade: um adversário que registre todo o tráfego cifrado de um usuário durante anos, esperando obter sua chave privada, encontrará ao obtê-la apenas um conjunto de chaves efêmeras já descartadas e irrecuperáveis. As mensagens históricas permanecem inacessíveis.

## 2.5. Segurança Pós-Comprometimento (*Post-Compromise Security*)

Complementar ao *Forward Secrecy*, a **Segurança Pós-Comprometimento** (PCS) endereça o cenário oposto: o que ocorre com as mensagens *futuras* após um comprometimento? Em protocolos sem essa propriedade, um atacante que obtenha as chaves ativas de uma sessão pode continuar lendo todas as mensagens subsequentes indefinidamente [Cohn-Gordon et al. 2020].

O Protocolo Signal resolve isso por meio da renovação contínua do material criptográfico a cada nova troca de mensagens. Mesmo que um atacante capture o estado completo de uma sessão em um dado instante, o protocolo “se cura” automaticamente nas próximas interações, tornando as mensagens futuras novamente opacas ao adversário. Essa é exatamente a propriedade que motivou o nome Axolotl, apresentada na Seção 1.

## 2.6. *Sealed Sender*

O *Sealed Sender* é uma propriedade presente no aplicativo Signal que vai além da proteção do conteúdo: ela oculta a identidade do **remetente** perante o próprio servidor. Em aplicativos de mensageria convencionais, o servidor precisa conhecer a origem de cada mensagem para roteá-la ao destinatário correto, gerando metadados de grafo social que revelam quem se comunica com quem. Com o *Sealed Sender*, essa informação é cifrada junto com a mensagem, de modo que o servidor entrega o pacote ao destinatário sem ter acesso à identidade de quem o enviou. A ausência dessa técnica no WhatsApp é um dos pontos centrais analisados na Seção 4.

# 3. O Protocolo Signal

O Protocolo Signal não é um algoritmo monolítico, mas uma orquestração sofisticada de primitivas criptográficas projetadas para garantir confidencialidade, integridade e autenticidade em ambientes assíncronos. Diferente de protocolos anteriores como o OTR (*Off-the-Record*), o Signal foi desenhado para funcionar em dispositivos móveis onde a conectividade é intermitente e múltiplos dispositivos podem estar associados a uma mesma identidade.

O funcionamento do protocolo pode ser dividido em três fases críticas: a infraestrutura de pré-chaves (registro), o estabelecimento de sessão (X3DH) e a renovação contínua de chaves (Double Ratchet).

### 3.1. Infraestrutura de Chaves (Pre-Keys)

Para permitir que Alice envie uma mensagem segura para Bob mesmo que ele esteja *offline*, o protocolo utiliza um modelo de publicação de chaves. Ao instalar o aplicativo, o cliente de Bob gera e envia ao servidor um conjunto de chaves públicas baseadas na curva elíptica Curve25519:

- **Identity Key ( $IK_B$ ):** Uma chave de longo prazo que identifica a conta de Bob (vinculada ao seu número de telefone).
- **Signed Pre-Key ( $SPK_B$ ):** Uma chave de médio prazo, assinada pela  $IK_B$  para garantir autenticidade. É renovada periodicamente.
- **One-Time Pre-Keys ( $OPK_B$ ):** Um lote de chaves efêmeras (ex: 100 chaves). O servidor entrega uma dessas chaves para cada nova solicitação de conversa e a remove do banco de dados imediatamente.

O servidor atua apenas como um diretório de distribuição (*Key Store*), sem nunca ter acesso às chaves privadas correspondentes.

### 3.2. Estabelecimento de Sessão: X3DH

O processo de "aperto de mão" inicial é denominado **X3DH** (*Extended Triple Diffie-Hellman*). Quando Alice deseja iniciar uma conversa, ela obtém o pacote de pré-chaves de Bob e realiza cálculos locais para derivar um segredo compartilhado ( $SK$ ).

O X3DH combina chaves de longo prazo e efêmeras para garantir autenticação mútua e sigilo encaminhado desde a primeira mensagem. O segredo  $SK$  é calculado através da concatenação dos resultados de quatro operações ECDH:

$$SK = KDF(DH1 \parallel DH2 \parallel DH3 \parallel DH4) \quad (3)$$

Onde os componentes são definidos como:

- $DH1 = DH(IK_A, SPK_B)$ : Autenticação mútua e identificação.
- $DH2 = DH(E_A, IK_B)$ : Autenticação da chave efêmera de Alice.
- $DH3 = DH(E_A, SPK_B)$ : Combinação de chaves efêmeras e assinadas.
- $DH4 = DH(E_A, OPK_B)$ : Garante o sigilo perfeito encaminhado inicial.

Alice então apaga sua chave efêmera privada ( $E_A$ ) e usa o  $SK$  para inicializar o algoritmo Double Ratchet. Ela envia sua chave pública efêmera junto com a primeira mensagem cifrada para que Bob possa reproduzir o cálculo.

### 3.3. O Algoritmo Double Ratchet

Uma vez estabelecida a sessão, o protocolo utiliza o algoritmo **Double Ratchet** para atualizar as chaves a cada mensagem. O nome "Catraca"(*Ratchet*) refere-se a um mecanismo que permite o avanço do estado criptográfico, mas impede sua reversão. O algoritmo combina dois tipos de catracas:

### 3.3.1. Ratchet Simétrico (Cadeia KDF)

Esta catraca avança a cada mensagem enviada ou recebida, garantindo que cada pacote de dados utilize uma chave única. O protocolo utiliza uma Função de Derivação de Chaves (KDF) baseada em HMAC-SHA256. A partir de uma *Chain Key* ( $CK_i$ ), o algoritmo deriva duas saídas: 1. Uma **Message Key** ( $MK_i$ ), usada exclusivamente para cifrar o conteúdo da mensagem atual. 2. A próxima **Chain Key** ( $CK_{i+1}$ ), que será usada na iteração seguinte.

$$MK_i = KDF_{mk}(CK_i) \quad \text{e} \quad CK_{i+1} = KDF_{ck}(CK_i) \quad (4)$$

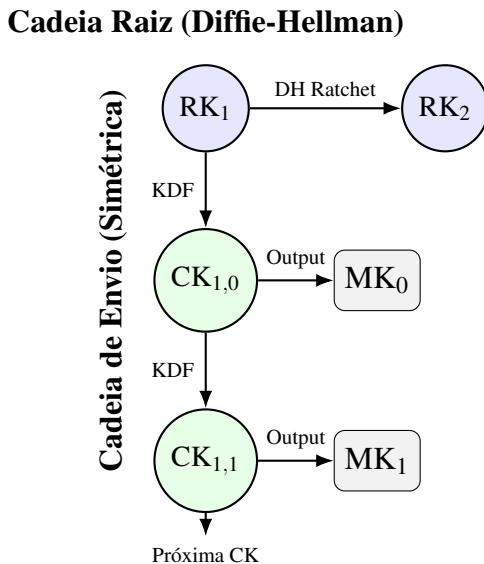
Como a KDF é uma função unidirecional, mesmo que um atacante obtenha a chave  $MK_i$ , ele não consegue calcular as chaves anteriores ( $MK_{i-1}$ ), garantindo o **Sigilo Perfeito Encaminhado**.

### 3.3.2. Ratchet Diffie-Hellman (Auto-Cura)

O Ratchet Simétrico protege o passado, mas é vulnerável se a *Chain Key* atual for roubada. Para mitigar isso, o protocolo executa um novo ECDH sempre que a conversa muda de direção (ex: Bob responde Alice).

As mensagens carregam novas chaves públicas efêmeras. Quando Bob responde, ele envia um novo valor DH público. Alice combina essa chave com sua chave privada para derivar uma nova "Raiz" de criptografia (*RootKey*). Isso reinicia as cadeias simétricas com entropia fresca, garantindo a **Segurança Pós-Comprometimento**.

A Figura 1 ilustra a interação entre a Raiz (Root Key) e as cadeias de envio.



**Figura 1. Representação do Double Ratchet.** A Cadeia Raiz (RK) é atualizada via Diffie-Hellman e alimenta a Cadeia de Envio (CK). A Cadeia de Envio gera chaves de mensagem (MK) individuais via KDF simétrico.

Em resumo, o Signal utiliza o X3DH para o acordo inicial e o Double Ratchet

para manter a sessão segura indefinidamente, "cicatrizando" a segurança da conversa automaticamente após qualquer comprometimento temporário.

## 4. Estudos de Caso e Implementações

[WESLEY] [explicar como o whatsapp usa o signal, o fato de ser código fechado impede de entrar em detalhes muito profundos, mas é essencial destacar a coleta de metadados do whatsapp, faça uma comparação antes e depois dos termos de uso de 2021, para a apresentação considere um exemplo prático como "se houvesse uma ordem judicial o que eles poderiam entregar? antes e depois" e de uma atenção especial para citar a fonte desses termos do whatsapp, pode ser o link do site deles...]

### 4.1. WhatsApp: Privacidade vs. Metadados

O WhatsApp é a implementação mais popular do Protocolo Signal. Embora o protocolo seja licenciado sob a GPLv3 (que exigiria que softwares derivados também fossem de código aberto), o WhatsApp mantém seu código fechado (proprietário).

Isso é possível porque o WhatsApp não simplesmente copiou o código do repositório público; eles firmaram um acordo comercial específico com a Open Whisper Systems em 2014 para integrar a biblioteca *libsignal* em sua infraestrutura antes que as obrigações estritas da GPL fossem aplicadas à sua base de código final [Marlinspike 2016].

#### 4.1.1. A Polêmica dos Metadados (2021)

Embora o conteúdo das mensagens no WhatsApp seja protegido pela mesma criptografia do Signal, o modelo de negócios da Meta (controladora do WhatsApp) baseia-se na exploração de **metadados**.

Em janeiro de 2021, o WhatsApp atualizou seus Termos de Serviço e Política de Privacidade, gerando reação global e migração em massa para o Signal e Telegram. A atualização explicitou a coleta de dados que não são protegidos pela criptografia ponta-a-ponta. Conforme a seção "Informações que coletamos" da política [WhatsApp Inc. 2021]:

- **Dados de Atividade:** Tempo, frequência e duração das interações.
- **Grafo Social:** Quem fala com quem (remetente e destinatário).
- **Informações do Dispositivo:** Modelo de hardware, sistema operacional, nível de bateria, força do sinal e identificadores únicos.
- **Endereço IP:** Usado para estimar a localização geral.

Ao contrário do aplicativo Signal, que desenhou seu sistema para minimizar o conhecimento do servidor (usando técnicas como *Sealed Sender*), o WhatsApp centraliza metadados, permitindo a construção de perfis comportamentais detalhados dos usuários, mesmo sem ler o conteúdo textual das mensagens.

### 4.2. Matrix e Olm/Megolm

[LA - WESLEY] [aqui é bom fazer uma menção honrosa para essa implementação promissora do signal pela Matrix, o olm é a implementação deles do protocolo signal como ele é, mas eles adicionam ao protocolo mais propriedades com o megolm, que serve para

garantir a mesma segurança (na pratica) para grupos gigantescos nível discord, mas hoje esta com a implementacao atrasada porque eles reescreveram todo o codigo deles em rust ha pouco tempo, mas vamos deixar isso no nosso texto mesmo que nao entre na apresentacao, e vamos resumir essa menção se nao der tempo de colocar diagram] O Matrix é um padrão aberto para comunicação descentralizada e federada. Diferente do Signal (que é centralizado), o Matrix permite que servidores diferentes conversem entre si. Para garantir segurança nesse ambiente, o Matrix desenvolveu a biblioteca **Olm**.

Olm é uma implementação do algoritmo Double Ratchet do Signal, escrita em C++ e adaptada para a arquitetura do Matrix. No entanto, o algoritmo original do Signal é "pesado" para grupos grandes (complexidade  $O(N^2)$ ), pois exige o estabelecimento de sessões individuais par-a-par.

Para resolver isso, o Matrix introduziu o **Megolm**. O Megolm sacrifica a propriedade de "auto-cura" imediata (o *break-in recovery* é mais lento) em troca de escalabilidade massiva para grupos com milhares de usuários. No Megolm, cada participante cria uma sessão de saída ("outbound") e compartilha a chave da catraca com os outros participantes via canais seguros Olm [The Matrix.org Foundation 2024]. Isso permite criptografia eficiente em ambientes federados sem quebrar as propriedades fundamentais de confidencialidade.

## 5. Conclusão

[TODOS][REVISAR para garantir consistencia e coesa] A evolução do algoritmo Axolotl para o onipresente Protocolo Signal redefiniu a segurança na internet. A tecnologia provou ser robusta contra adversários estatais e corporativos no que tange à confidencialidade do conteúdo.

Entretanto, a análise das implementações do WhatsApp e Matrix revela que o protocolo é apenas uma peça do quebra-cabeça. No caso do WhatsApp, a segurança criptográfica convive com a vigilância de metadados sancionada pelos termos de uso. No caso do Matrix, vemos a adaptação do protocolo (Megolm) para garantir soberania de dados através da federação. Conclui-se que a "privacidade" é um conceito mais amplo que a "criptografia", dependendo tanto da matemática dos algoritmos quanto da ética das implementações.

## Referências

- Antunes, I. and Kowada, L. A. (2018). Explorando o sistema de criptografia Signal no WhatsApp. In *Anais do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)*. SBC.
- Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., and Stebila, D. (2020). A formal security analysis of the signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983.
- Marlinspike, M. (2016). Whatsapp's signal protocol integration. <https://signal.org/blog/whatsapp-complete/>.
- Perrin, T. and Marlinspike, M. (2016). The double ratchet algorithm. Technical report, Signal.

The Matrix.org Foundation (2024). Matrix Specification - E2E Encryption Implementation. <https://spec.matrix.org/latest/client-server-api/#end-to-end-encryption>. Descrição das bibliotecas Olm e Megolm.

WhatsApp Inc. (2021). WhatsApp Privacy Policy Updates. <https://www.whatsapp.com/legal/updates/privacy-policy>. Termos atualizados em Janeiro de 2021.