UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉ GUSTAVO LIMA FIGUEIREDO
VANESSA ESCÓCIO LOURENÇO

ANÁLISE DO MODELO DE ATAQUES AO PROTOCOLO SIGNAL

Uma abordagem usando lógica e mCRL2

RIO DE JANEIRO
2020

ANDRÉ GUSTAVO LIMA FIGUEIREDO
VANESSA ESCÓCIO LOURENÇO

ANÁLISE DO MODELO DE ATAQUES AO PROTOCOLO SIGNAL

Uma abordagem usando lógica e mCRL2

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Mário Roberto Folhadela Benevides
Co-orientador: Profa. Valeria M. Bastos

RIO DE JANEIRO
2020

Figueiredo, André Gustavo Lima
F475a Análise do modelo de ataques ao protocolo Signal: uma abordagem usando lógica e mCRL2 / André Gustavo Lima Figueiredo, Vanessa Escócio Lourenço. – 2020.

59 f.

Orientador: Mário Roberto Folhadela Benevides.
Coorientadora: Valéria Menezes Bastos.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2020.

1. Signal. 2. Criptografia ponta-a-ponta. 3. Verificação. 4. mCRL2. I. Lourenço, Vanessa Escócio. II. Benevides, Mário Roberto Folhadela (Orient.). III. Bastos, Valéria Menezes (Coorient). IV. Universidade Federal do Rio de Janeiro, Instituto de Matemática. V. Título.

ANDRÉ GUSTAVO LIMA FIGUEIREDO
VANESSA ESCÓCIO LOURENÇO

ANÁLISE DO MODELO DE ATAQUES AO PROTOCOLO SIGNAL
Uma abordagem usando lógica e mCRL2

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 16 de Junho de 2020

BANCA EXAMINADORA:

Prof. Mário R. Folhadela Benevides
Ph.D

Profa. Valeria Menezes Bastos
D.Sc

Prof. Claudio Miceli de Farias
D.Sc

AGRADECIMENTOS

A Deus, pela força e consolo. A família, amigos e a todos professores que nos ajudaram ao longo da graduação. Ao professor Mário R. Folhadela Benevides pela paciência e mentoria na realização desse trabalho.

RESUMO

O objetivo deste trabalho foi analisar possíveis ataques ao protocolo *Signal* através de modelagens utilizando a ferramenta mCRL2. O protocolo *Signal* é constituído pelos protocolos *Extended Triple Diffie-Hellman* e o *Double Ratchet*, ambos criados em 2013. O *Signal*, um dos primeiros protocolos a implementar criptografia ponta-a-ponta, é amplamente utilizado nos principais aplicativos de troca de mensagens, como o *Whatsapp*. Devido à sua popularidade, o protocolo foi escolhido neste trabalho para verificarmos se ele é realmente seguro, ou seja, se um invasor pode interceptar todas as mensagens trocadas entre dois usuários. A ferramenta mCRL2 permitiu a especificação, visualização, simulação e verificação formal do protocolo. Neste trabalho discutimos algumas estratégias de interceptações conhecidas, juntamente com uma análise dos resultados obtidos com o uso da ferramenta mCRL2.

Palavras-chave: Signal. Criptografia ponta-a-ponta. Verificação. mCRL2.

ABSTRACT

The aim of this study was to analyze possible attacks on the Signal protocol through models using the mCRL2 tool. The Signal protocol is made up of Extended Triple Diffie-Hellman and Double Ratchet protocols, both created in 2013. Signal, one of the first protocols to implement end-to-end encryption, is widely used in the main messaging applications, such as WhatsApp. Due to its popularity, the protocol was chosen in this work to check if it is really safe, that is, if an attacker can intercept all messages exchanged between two users. The mCRL2 tool allowed the specification, visualization, simulation and formal verification of the protocol. In this work, we discuss some known interception strategies, as well as an analysis of the results of the mCRL2 tool.

Keywords: Signal. End to end cryptography. Verification. mCRL2.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GDPR	General Data Protection Regulation
LGPD	Lei Geral de Proteção de Dados
SIM	Subscriber Identification Module
CA	Certificate Authorities
MiTM	Man in the Middle
SHA-1	Secure Hash Algorithm 1
OTR	Off-the-Record Messaging
AES	Advanced Encryption Standard
IK	Identity Key
SPK	Signed Prekey
OPK	One-time prekey
EK	Ephemeral Key
AD	Associate Data
AEAD	Authenticated Encryption with Associated Data
DH	Diffie-Hellman
SK	Shared Key
PFS	Perfect Forward Secrecy
KDF	Key Diffie-Hellman Function
X3DH	Extended Triple Diffie-Hellman
dhout	Diffie-Hellman Output Key
UKS	Unknown Key-Share Attack
TOFU	Trust on First Use
PKI	Public Key Infrastructure
LTS	Labelled Transitions System

LPS	Linear Process Specifications
PBES	Parameterised Boolean Equation Systems
GPL	General Public License
A	Ana
B	Beto
C	Carlos
Ea	Chave de encriptação de Ana
Eb	Chave de encriptação de Beto
Da	Chave de decriptação de Ana
Db	Chave de decriptação de Beto
M	Mensagem em texto claro
Mc	Mensagem criptografada

SUMÁRIO

1 INTRODUÇÃO

O objetivo deste trabalho é analisar, especificar, visualizar e testar possíveis problemas no protocolo *Signal*. Por meio de uma série de artefatos que serão apresentados ao longo dos capítulos, mostraremos um modelo de ataque que poderia ser bem sucedido se algumas precauções não fossem tomadas.

A motivação do projeto se baseou no aplicativo que teve o maior número de downloads no Brasil e no mundo nos últimos anos , o Whatsapp (J.CLEMENT, 2019). O protocolo *Signal* é utilizado para criptografar as mensagens enviadas via Whatsapp. Em razão da popularidade do aplicativo, ele também tem sido alvo de muitos ciberataques e polêmicas devido a sua criptografia fim-a-fim, que serão mencionadas nesse trabalho.

Os crimes cibernéticos e fraudes estão cada vez mais comuns no Brasil devido a alta lucratividade e desinformação de boas práticas de segurança por parte dos usuários. Um exemplo de golpe comum é a vítima atender um telefonema diretamente de um aplicativo de comunicação e o suspeito pedir para que ela abra um link para o código secreto da aplicação e que repasse esse código para ele. A partir desse momento o criminoso consegue acessar os contatos da vítima e inicia as extorsões, conforme aconteceu com alguns artistas nacionais (R7, 2020).

O ataque chamado *SIM Swap*, que consiste na transferência da linha do celular para um novo chip SIM, seria um tipo de ataque bem sucedido se o protocolo guardasse as mensagens criptografadas no servidor. Como a maioria dos aplicativos de troca de mensagens pelo celular utiliza o número associado ao chip SIM como identificador da conta, o invasor consegue acesso a possíveis mensagens guardadas no servidor, já que a principal referência de identidade é o número associado ao chip.

Esse problema não ocorre com o *Signal* e qualquer outro aplicativo que use a criptografia fim-a-fim do protocolo. Um exemplo de aplicativo com esse problema seria o Telegram que, em meados de 2019, ganhou evidência no Brasil devido a problemas com o vazamento de mensagens trocadas por pessoas do alto escalão do governo e da justiça brasileira (Greenwald et al., 2019).

No capítulo 2 é apresentado o protocolo *Signal* com detalhes de suas características contendo a descrição dos tipos de chaves empregadas no início da sessão, o funcionamento da troca de mensagens, protocolos atrelados e aspectos importantes na área da Segurança da Informação.

No capítulo 3 é discutido o modelo Dolev Yao, um dos primeiros modelos formais de segurança de protocolos criptográficos assimétricos. Além disso, são abordados temas como Autoridades Certificadoras (CA) e a dificuldade de identificar um usuário na Internet. No mesmo capítulo, na Seção 3.2, discutimos um modelo de ataque chamado *Unknown Key Share*. Nesse ataque, o invasor finge ser a pessoa da outra ponta da conversa e consegue

interceptar as mensagens trocadas.

No capítulo 4 é apresentada uma linguagem de especificação chamada mCRL2 (GROOTE et al., 2007) que tem a finalidade de analisar e permitir a visualização de comportamentos, processos e protocolos. Ao longo do capítulo também são definidas as especificações do protocolo *Signal* através da utilização da linguagem mCRL2, bem como uma pequena simulação de troca de chaves entre Ana e Beto, e interceptação de um invasor. Carlos, o agente malicioso, utiliza um tipo de ataque que consiste em se colocar no meio do caminho da comunicação entre Ana e Beto. A esse tipo de ataque se dá o nome de *Man in the Middle*, ou simplesmente MiTM (CONTI; DRAGONI; LESYK, 2016).

E, por fim, no capítulo 5, são apresentados comentários e discutidas sugestões de trabalhos futuros, que motivam o uso de ferramentas mais complexas como o Proverif (BLANCHET et al., 2018) para verificação de segurança de protocolos criptográficos usando cláusulas de Horn e o modelo Dolev Yao.

1.1 TRABALHOS CORRELATOS

Este estudo teve como inspiração o artigo *"A Formal Security Analysis of the Signal Messaging Protocol"* (COHN-GORDON et al., 2017), que desde 2017 vem recebendo incrementos e atualizações em seu texto, tornando-o mais robusto e completo. Além de ser um dos pioneiros no assunto. Em sua última versão, datada de Julho de 2019, o artigo destrincha o protocolo *Signal* buscando traduzir cada etapa dele em sentenças lógicas, com o objetivo de realizar análises complexas sobre sua segurança.

Nosso trabalho não tem como objetivo provar se o *Signal* é seguro ou não, pois isso pode ser demasiadamente complexo para um trabalho de graduação e pode nos levar a caminhos e conclusões incorretas sobre temas sensíveis, como a eficácia do protocolo. Nosso objetivo foi analisá-lo sob o prisma de modelos de ataques já existentes, como o *Dolev Yao* (DOLEV; YAO, 1983) e o *UKS (Unknown Key Share attack* (BERNSTEIN, 1999), e apresentar possíveis problemas e pontos fracos.

Como nosso objetivo era utilizar a linguagem e ferramentas do mCRL2 (GROOTE et al., 2007), nós buscamos na literatura acadêmica, trabalhos que utilizassem a mesma abordagem, mas para protocolos diferentes do *Signal*. O trabalho que mais se aproximou da nossa proposta e serviu como apoio foi o *"Modelling and analysing a WSN secure aggregation protocol: A comparison of languages and tool support"* (CAMBAZOGLU et al., 2015). O artigo utiliza o mCRL2 para modelar o protocolo chamado *WSN (Wireless Sensor Networks*. Os autores simulam um invasor na rede de sensores, seguindo os mesmos princípios apresentados por *Dolev Yao* (DOLEV; YAO, 1983), que é semelhante ao que fizemos com o protocolo *Signal*. Nosso trabalho é um dos pioneiros a apresentar uma abordagem envolvendo o mCRL2 e o protocolo *Signal*, portanto nosso trabalho não é uma mera reprodução desse estudo.

Por fim, em nossas pesquisas encontramos estudos bem promissores sobre a abordagem lógica utilizando o modelo do atacante *Dolev Yao* (DOLEV; YAO, 1983) em protocolos de comunicação. Um projeto que podemos citar é o Proverif (BLANCHET et al., 2018), que criou uma própria abordagem para testar protocolos criptográficos mas que não foi abordado nesse trabalho pois já tínhamos iniciado usando as ferramentas do mCRL2.

2 CAPÍTULO 2 - O PROTOCOLO SIGNAL

O Protocolo *Signal*, também conhecido como *TextSecure Protocol*, foi desenvolvido em 2013 por Trevor Perrin e Moxie Marlinspike (MARLINSPIKE, 2014), fundadores da empresa *Open Whisper Systems*. A primeira versão do protocolo, chamada de TextSecure v1, foi baseada em outro protocolo chamado *Off-the-Record Messaging* (OTR) (STEDMAN, 2008). O OTR usa uma combinação do AES (*Advanced Encryption Standard*) (FIPS, 2001) com o algoritmo de troca de chaves Diffie-Hellman (HELLMAN, 1976) e a função de hash SHA-1 (D. Eastlake 3rd, Motorola, P. Jones, Cisco Systems, 2001) para a troca segura de mensagens.

Em Fevereiro de 2014, a mesma empresa lançou o TextSecure v2. A novidade foi o uso do algoritmo *Ratchet* Duplo (PERRIN, 2016), que possibilitou o suporte a troca de mensagens de maneira assíncrona, bem comum nos aplicativos de troca de mensagens hoje em dia.

Mais tarde, mais uma versão do protocolo foi lançada: o TextSecure v3, que em Novembro de 2015 foi renomeado para Protocolo *Signal* (Moxie Marlinspike, 2015), logo depois de ser incorporado a outro aplicativo, chamado RedPhone. Nessa terceira versão e na subsequente foram implementadas melhorias na criptografia e na transmissão de dados fim a fim.

Em 21 de fevereiro de 2018 *Moxie* publicou no blog oficial da solução a seguinte notícia sobre o *Signal Foundation* (Moxie Marlinspike, 2018):

> *We're glad those are the choices we've made. Today, we are launching the Signal Foundation, an emerging 501(c)(3) nonprofit created and made possible by Brian Acton, the co-founder of WhatsApp, to support, accelerate, and broaden Signal's mission of making private communication accessible and ubiquitous. In case you missed it, Brian left WhatsApp and Facebook last year, and has been thinking about how to best focus his future time and energy on building nonprofit technology for public good.*
>
> *Starting with an initial* $\$50,000,000$ *in funding, we can now increase the size of our team, our capacity, and our ambitions. This means reduced uncertainty on the path to sustainability, and the strengthening of our long-term goals and values. Perhaps most significantly, the addition of Brian brings an incredibly talented engineer and visionary with decades of experience building successful products to our team.*

Como o próprio *Moxie* informa na notícia, isso possibilitou o *Signal* investir no aumento do time e na qualidade dos programadores para desenvolver mais recursos para o protocolo e para o aplicativo, que hoje não tem tantos recursos de mídia quanto os seus concorrentes: Whatsapp e Telegram, por exemplo. Em 27 de novembro de 2019 foi anunciada a versão 3 do *Signal* para iPad, além de melhorias para as plataformas iOS. Essas

atualizações tendem a ficar cada vez mais frequentes a medida que o protocolo se expande para outras plataformas de troca de mensagem e, claro, permaneça com o compromisso em manter a privacidade dos seus usuários que é sempre relembrado nas publicações do blog oficial, como este de Outubro de 2018. (Joshua Lund, 2018):

> *In addition to the end-to-end encryption that protects every Signal message, the Signal service is designed to minimize the data that is retained about Signal users. By design, it does not store a record of your contacts, social graph, conversation list, location, user avatar, user profile name, group memberships, group titles, or group avatars.*
>
> *We have been exploring techniques to further reduce the amount of information that is accessible to the service, and the latest beta release includes changes designed to move Signal incrementally closer to the goal of hiding another piece of metadata: who is messaging whom.*

2.1 TIPOS DE CHAVES USADAS NO PROTOCOLO SIGNAL

Em tempo de instalação do aplicativo, cada agente gera uma série de chaves que serão usadas para troca de informações e mensagens. Duas delas são as chaves de longo prazo que são geradas cada vez que o aplicativo é instalado ou reinstalado. Também temos as chaves de médio prazo e as chaves de uso único que são usadas uma única vez. Abaixo descreveremos os quatro tipos de chaves usadas no protocolo.

Primeiro temos as chaves de identidade, que vamos chamar de IK (*Identity Key*). Elas são geradas em tempo de instalação do aplicativo e definem a identidade do usuário.

Depois de geradas as chaves de identidades, ainda em tempo de instalação, é gerada a chave assinada. Ela é assinada com a chave de identidade que foi gerada anteriormente e é recriada de tempos em tempos, por isso é uma chave de médio prazo. Vamos chamá-la pela abreviatura: SPK (*Signed Prekey*). A assinatura para assinar essa chave é gerada da seguinte maneira: *Signature (IK, Encode (SPK))*, ou seja, assinatura gerada usando a chave de identidade em conjunto com a chave assinada codificada da outra parte. Antes de iniciar a comunicação o usuário solicita ao outro participante a assinatura e realiza o processo de verificação de autenticidade. Caso seja confirmada a identidade o processo pode continuar. Caso negativo, o processo é abortado.

Também temos as chaves de uso único, que chamaremos pela abreviatura: OPK (*One-time prekey*). Essas chaves são geradas em lotes, de tempos em tempos, e enviadas para os servidores centrais da aplicação. Cada uma dessas chaves é usada somente uma vez para a troca de uma única mensagem e depois é apagada. Quando todas as chaves desse tipo são consumidas do servidor, novas chaves são geradas pelo cliente que acabam por abastecer o servidor novamente. Elas são também do tipo Curve25519 (BERNSTEIN, 2006), uma curva elíptica usada para gerar chaves no algoritmo Diffie-Hellman (HELLMAN, 1976).

Há ainda um outro tipo de par de chaves criada sempre que alguém quiser iniciar a troca de mensagem com alguém. No nosso texto, exceto quando especificado o contrário,

Ana é o agente não malicioso que sempre iniciará a troca de mensagens. A esse tipo de chave daremos o nome de EK (*Ephemeral Key*) pois ela é gerada, usada uma vez no início da comunicação e depois descartada. Abaixo temos uma tabela descrevendo as chaves usadas no protocolo. Na tabela 1 também definimos a notação que identifica o dono de cada chave.

Tabela 1 – Tabela com descrição das chaves usadas no Signal

NOME	DEFINIÇÃO
IK_A	Chave de identidade da Ana
EK_A	Chave Efêmera da Ana
IK_B	Chave de identidade do Beto
SPK_B	Chave assinada do Beto
OPK_B^n	Chaves de uso único do Beto

Fonte: https://signal.org/docs/specifications/x3dh/ (2016)

2.2 O REGISTRO DO CLIENTE E O INÍCIO DA SESSÃO

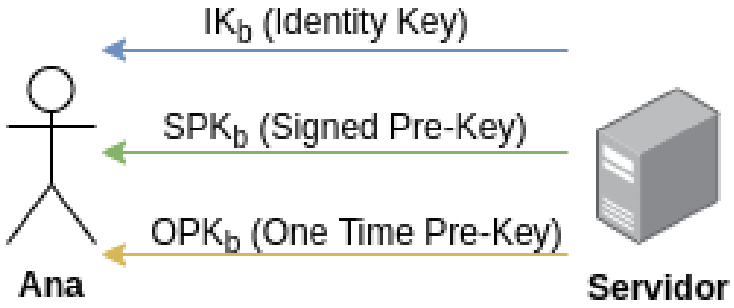
Cada agente precisa se comunicar com um servidor central para disponibilizar suas chaves públicas: IK (*Identity Key*) para a identificação do usuário, SPK (*Signed Pre-Key*) pré-assinada pela IK e suas OPK (*One-Time prekey*) de uso único.

Esse envio é realizado no tempo de registro, portanto essa fase chama-se registro do cliente. Tanto Ana quanto Beto executam esse passo. Em momento algum o servidor guarda a chave privada das partes ou qualquer outra informação que possa ser usada para decifrar as mensagens futuramente.

Finalizado o registro das partes, com o envio das devidas chaves ao servidor, é possível iniciar uma sessão entre elas. Se Ana quer comunicar-se com Beto, ela requisita ao servidor as chaves públicas: IK_B, SPK_B e uma chave de uso único, por exemplo, a OPK_B^1, mostrada na figura 1. Lembrando que as chaves efêmeras são usadas uma única vez, então imediatamente essa chave OPK_B^1 do Beto é removida do servidor. Se não existir uma chave efêmera desse tipo no servidor, nenhuma chave desse tipo é enviada. Se isso ocorrer, significa que Beto ainda não atualizou o servidor com um novo conjunto de chaves e deve fazer isso o quanto antes. Esse fato não inviabiliza a comunicação, pois a chave de uso único é opcional na troca de mensagens.

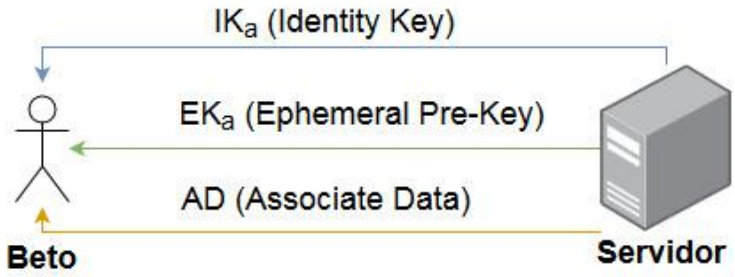
Em posse dessas chaves, Ana primeiro verifica a chave assinada do Beto, se o resultado for negativo ela aborta a comunicação, caso contrário ela configura o início da sessão de longa duração com Beto. Para isso ela usa sua EK_A em conjunto com todas as chaves do Beto, enviadas pelo servidor, como entrada em uma função chamada KDF (KRAWCZYK; ERONEN, 2010), que tem como saída uma *Shared Key*. Na tabela 2

Figura 1 – Ana solicitando chaves de Beto



Fonte: Autoria própria

Figura 2 – Beto solicitando chaves de Ana



Fonte: Autoria própria

abaixo nós esquematizamos como é feita a entrada na função HKDF, no caso em que não haja OPK_B e para o caso que exista a OPK_B, na tabela 3:

Tabela 2 – Tabela com operações quando falta a OPK_B

RESULTADO	OPERAÇÃO
DH_1	DH (IK_A, SPK_B)
DH_2	DH (EK_A, IK_B)
DH_3	DH (EK_A, SPK_B)
SK	KDF (DH_1 || DH_2 || DH_3)

Fonte: https://signal.org/docs/specifications/x3dh/ (2016)

Depois de Ana calcular a *Shared Key* (SK), ela ainda calcula uma sequência de bytes de "dados associados"(denominado AD), contendo informações sobre identidade tanto dela quanto de Beto. Na tabela 4 podemos ver as informações contidas em AD.

Depois de criado esses dados associados e a *Shared Key*, Ana envia a Beto uma mensagem inicial contendo: A sua chave de identidade (IK_A), a sua chave efêmera (EK_A), um identificador informando qual prekey de Beto foi usada e um texto cifrado com alguma

Tabela 3 – Tabela com operações quando existe a OPK_B

RESULTADO	OPERAÇÃO
DH_1	DH (IK_A, SPK_B)
DH_2	DH (EK_A, IK_B)
DH_3	DH (EK_A, SPK_B)
DH_4	DH (EK_A, OPK_B)
SK	KDF (DH_1 || DH_2 || DH_3 || DH_4)

Fonte: https://signal.org/docs/specifications/x3dh/ (2016)

Tabela 4 – Cálculo dos dados associados (AD)

RESULTADO	OPERAÇÃO
AD	Encode (IK_A) || Enconde (IK_B)

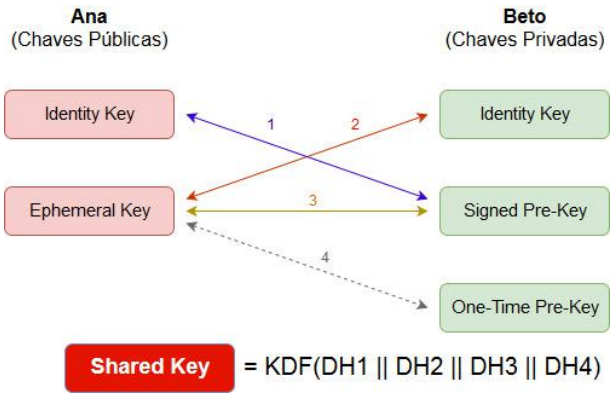
Fonte: https://signal.org/docs/specifications/x3dh/ (2016)

método de criptografia AEAD (ROGAWAY, 2002), usando o AD como dados associados e a *Shared Key* gerada anteriormente.

Essa mensagem inicial serve para duas coisas: serve para marcar a primeira mensagem no protocolo $X3DH$ e a primeira mensagem $X3DH$ de Ana. Depois dela, Ana pode continuar usando a *Secret Key* ou derivar novas chaves para se comunicar com Beto.

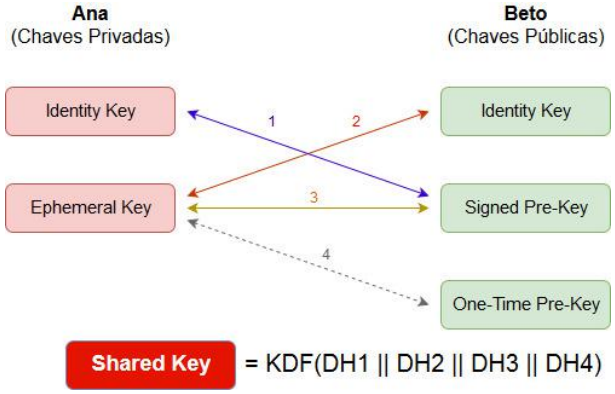
Depois de receber a mensagem inicial de Ana, Beto recupera a chave de identidade (IK_A) e a chave efêmera (EK_A) dela. Beto usa sua chave de identidade privada, sua chave privada da prekey e a chave privada referente a *one-time prekey* usadas por Ana para conseguir verificar o conteúdo da mensagem. Abaixo, na figura 4, temos o esquema gráfico de formação do segredo compartilhado (SK) de Ana e na figura 3 temos o esquema gráfico para formação de chave de Beto.

Figura 3 – Esquema gráfico de formação de segredo compartilhado no lado de Beto



Fonte: Autoria própria

Figura 4 – Esquema gráfico de formação de segredo compartilhado no lado de Ana



Fonte: Autoria própria

2.3 O DOUBLE RATCHET E O CONTROLE DE SESSÃO

O controle de sessão se dá através do algoritmo *Double Ratchet*, criado por Trevor Perrin e Moxie Marlinspike (PERRIN, 2016), em 2013. O algoritmo X3DH (Seção 2.2) é o protocolo acordado entre as partes para troca de chaves que serão usadas e o *Double Ratchet* é o protocolo usado para trocar mensagens criptografadas, além de manutenção da segurança da sessão. Depois de ambas as partes possuírem todas as chaves e conferirem a autenticidade uma da outra, ou seja a sessão foi iniciada, é necessário fazer o controle dela. O *Double Ratchet* entra em cena para controlar a sessão e a troca de mensagens criptografadas entre as partes.

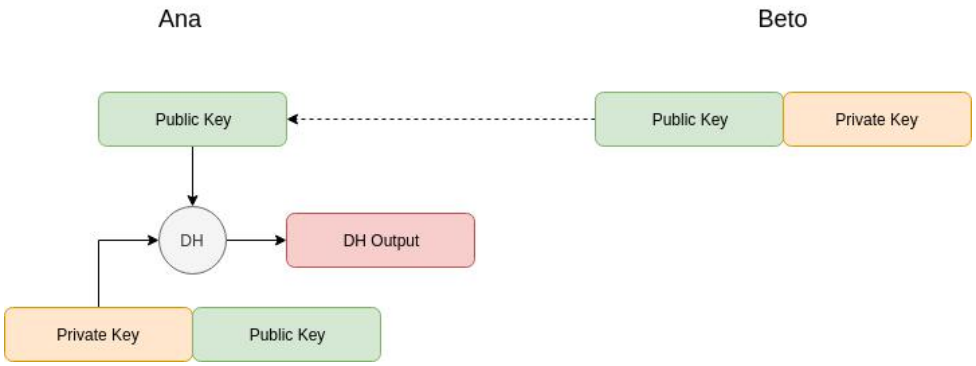
O grande desafio dos aplicativos de troca de mensagens é a assincronicidade. Para se manter a qualidade de *Perfect Forward Secrecy* (KRAWCZYK, 2011) é necessário haver trocas constantes de chaves efêmeras entre as partes. Entretanto, no *Signal* e em qualquer aplicativo de troca de mensagens, um dos participantes pode estar offline e não ser capaz de realizar essa troca com frequência. O *Double Ratchet* resolve esse problema.

Vamos supor que Ana esteja iniciando o protocolo. Ela começa gerando um par de chaves efêmeras DHK_A^1 e solicita uma chave pública efêmera DHK_B^1 a Beto. Ela usa a sua chave privada do par gerado e junta com a pública, enviada por Beto, como entrada numa função Diffie-Hellman (STEINER; TSUDIK; WAIDNER, 1996), que não entraremos em detalhes pois não faz parte do escopo do trabalho, mas consideraremos ela segura o suficiente. Essa função tem como saída uma chave que chamaremos de "dhoutk"(*Diffie-Hellman Output Key*), como mostrado na Figura 5.

Do lado do Beto,Figura 6, a correspondente chave privada do par dele e a chave pública de Ana serão usadas como entrada na mesma função Diffie-Hellman (STEINER; TSUDIK; WAIDNER, 1996), tendo como saída a mesma "dhoutk"encontrada por Ana. E na figura 7 temos o esquema gráfico de alguns passos do Ratchet:

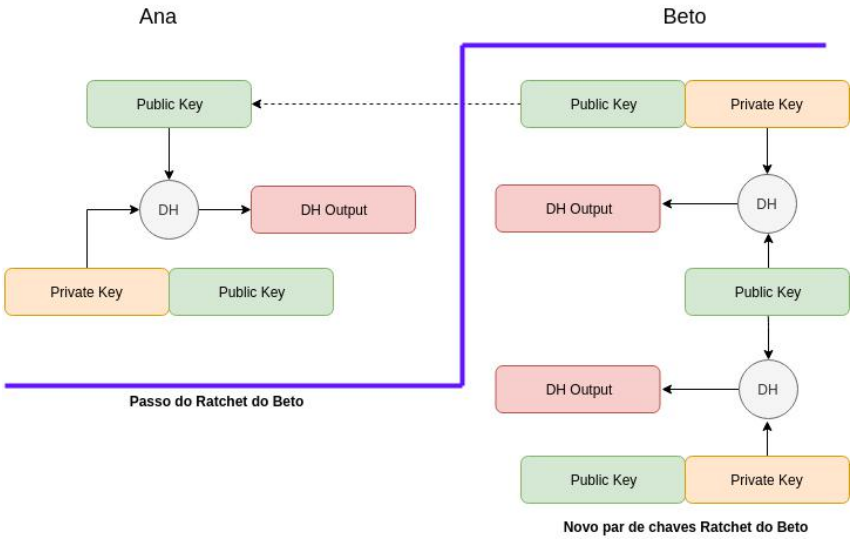
Na figura 7, por exemplo, temos Ana requisitando uma chave pública do Beto e uti-

Figura 5 – Primeiro passo do Ratchet



Fonte: https://signal.org/docs/specifications/doubleratchet/ (2016)

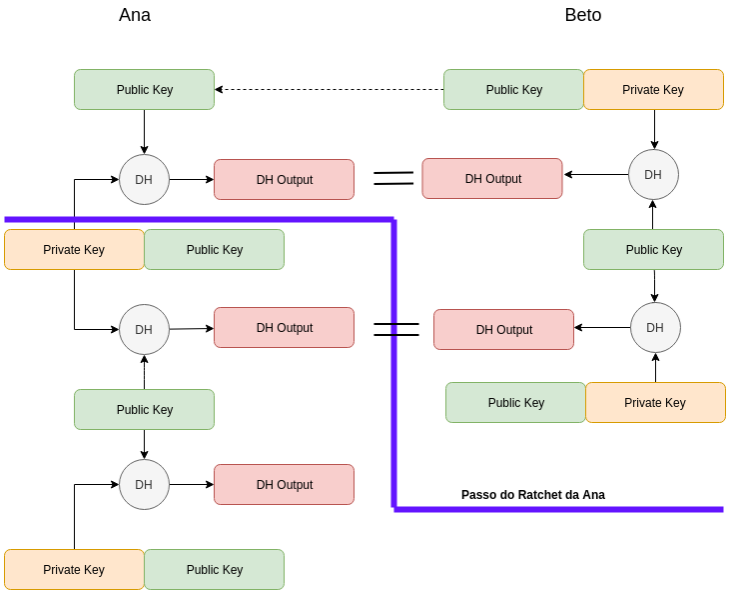
Figura 6 – Primeiro e segundo passo do Ratchet



Fonte: https://signal.org/docs/specifications/doubleratchet/ (2016)

lizando sua chave privada como entrada numa função, obtendo como saída um segredo compartilhado, denominado na figura como *DH Output*. Essa chave só será usada para a troca dessa mensagem específica. Note que a linha em azul separa a segunda troca de mensagens. Ana aproveita a chave privada do passo anterior, mas solicita mais uma chave pública a Beto para gerar mais um outro segredo compartilhado. Podemos notar que depois há geração de mais um segredo compartilhado para a troca de uma outra mensagem. O *Signal*, portanto, gera uma chave diferente para cada troca de mensagem entre as partes. Se alguém interceptando o tráfego tiver acesso as mensagens criptografadas para tentar quebrá-las no futuro, o atacante terá que quebrar todos esses segredos compartilhados gerados em cada troca de mensagem, do contrário não conseguirá quebrar as mensagens cifradas.

Figura 7 – Primeiro, segundo e terceiro passo do Ratchet



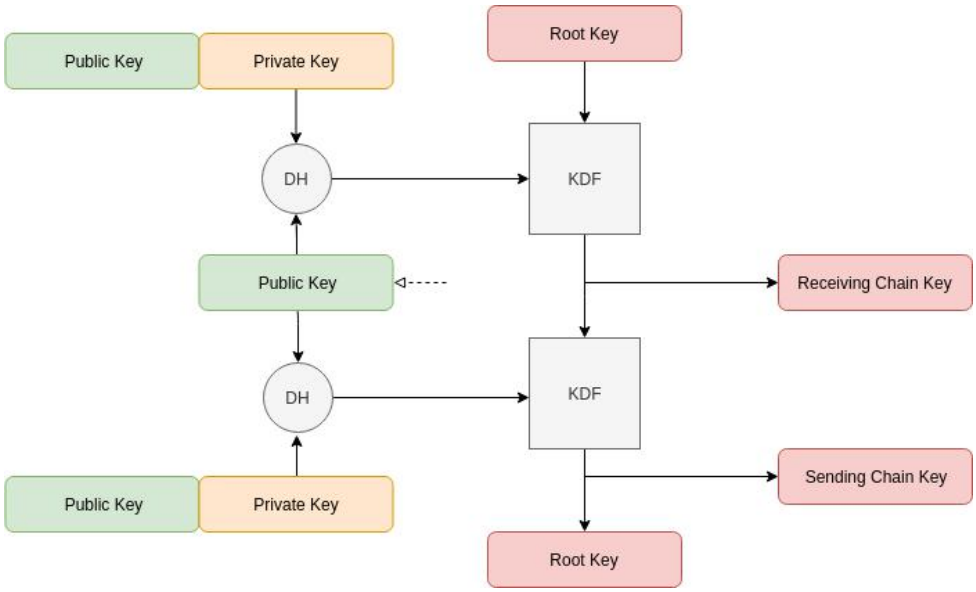
Fonte: https://signal.org/docs/specifications/doubleratchet/ (2016)

As figuras 5, 6 e 7 são uma simplificação do algoritmo *Diffie-Hellman Ratchet*. É preciso ainda especificar que a chave de saída da função Diffie-Hellman é usada como entrada em outra função chamada KDF (KRAWCZYK; ERONEN, 2010), que também não entraremos em detalhes, mas que é considerada matematicamente segura. A figura 8 mostra a parte simétrica do algoritmo. O *Diffie-Hellman Ratchet* em conjunto com o passo simétrico do *Ratchet* compõe o algoritmo de *Ratchet* duplo.

Podemos notar que o segredo compartilhado não é gerado simplesmente da saída da função DH como especificado nas figuras 5, 6 e 7. A saída da função é usada como entrada na função KDF, junto com a Root Key e, a partir daí temos uma outra saída, denominada *Receiving* ou *Sending Chain Key*, depende da função dela no momento de troca de mensagem. Essa *Chain Key* ainda é usada novamente na função KFD para gerar a chave de recebimento ou a chave de envio, como mostrado na figura 9.

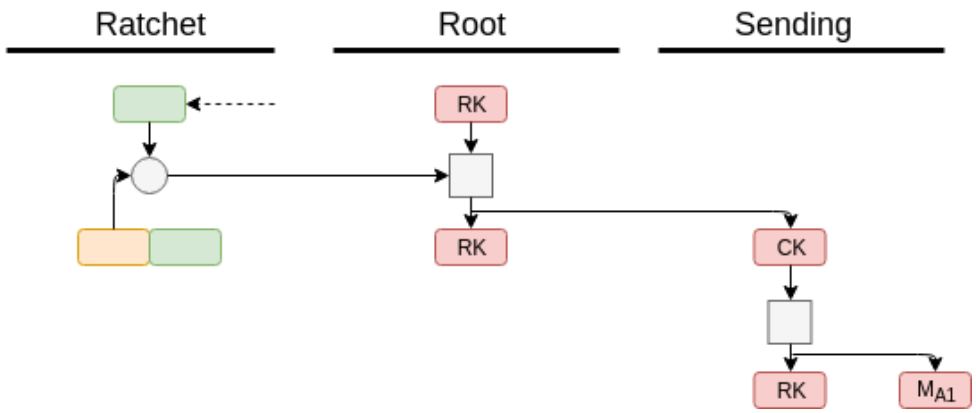
Supondo que Ana queira enviar uma mensagem M_{A1} para Beto. Ela realiza os passos assimétricos e simétricos, resultando na mensagem criptografada M_{A1}. Essa mensagem foi criptografada com uma chave de envio. A primeira usada por Ana. Vale ressaltar que, como futuras mensagens serão sempre criptografadas com outras chaves, todas as antigas chaves geradas no processo, como as: *Chain Keys* e *Root Keys*, podem ser removidas. Nas especificações do protocolo são indicadas que algumas chaves serão mantidas por algum tempo para o caso de ser necessário pegar uma mensagem que tenha chegado atrasada. Mensagens muito antigas e que não puderam ser entregues antes da destruição da chave, não podem mais ser lidas, nem mesmo pelos participantes da conversa.

Figura 8 – Ratchet: passo simétrico junto com o assimétrico



Fonte: https://signal.org/docs/specifications/doubleratchet/ (2016)

Figura 9 – Ratchet: Finalmente a chave usada para trocar a mensagem



Fonte: https://signal.org/docs/specifications/doubleratchet/ (2016)

Depois de receber a mensagem de Ana, vamos supor que Beto queira enviar uma resposta M_{B1}. Ele envia uma outra chave pública para Ana formar a *Receiving Key* daquele passo, ao mesmo tempo que gera uma nova *Sending Key* com uma chave pública de Ana. Para resposta, portanto, ele gera uma nova chave e criptografa a mensagem de resposta com ela.

3 CAPÍTULO 3 - MODELOS DE VERIFICAÇÃO DE SEGURANÇA DE PROTOCOLOS CRIPTOGRÁFICOS ASSIMÉTRICOS

3.1 O MODELO DOLEV YAO

Um dos primeiros modelos formais de verificação de segurança de protocolos criptográficos assimétricos foi o modelo Dolev Yao (DOLEV; YAO, 1983). E para que um sistema de chaves públicas funcione perfeitamente, primeiro temos que assumir algumas premissas: as funções de uma via (funções *hash*) são suficientemente confiáveis; o local que abriga as chaves públicas precisa ser seguro e não podem ser modificados por alguém sem autorização; o repositório de chaves públicas precisa ser público e todos precisam ter acesso para consultar as chaves dos outros agentes com quem pretendem se comunicar; somente o agente gerador das chaves possui acesso a chave privada correspondente.

Além disso, num protocolo de comunicação entre duas partes, somente os dois participantes encriptam e decriptam as mensagens trocadas. No caso do protocolo do *Signal*, o servidor só serve como repositório de chaves públicas e chaves públicas efêmeras. O servidor não participa diretamente dos processos.

Dolev e Yao assumem também o seguinte sobre o atacante:

1. O atacante pode analisar e capturar todas as mensagens trocadas na rede, inclusive as mensagens contendo as chaves;

2. O atacante tem a capacidade de ler, alterar, bloquear e redirecionar todas as mensagens enviadas no protocolo. Além disso, ele é capaz de inserir novas mensagens no meio da comunicação;

3. O atacante pode ser um legítimo participante do protocolo (insider), um participante externo ou uma combinação dos dois;

4. O atacante pode iniciar um número incontável de rodadas do protocolo em paralelo entre qualquer agente. Isso inclui diferentes rodadas com diferentes agentes. Ou seja, o atacante pode estar numa determinada rodada com um agente específico e em outra rodada com outro(s) agente(s).

3.2 EXEMPLO DE ATAQUE A UM PROTOCOLO ASSIMÉTRICO SIMPLES

Para ilustrar o modelo de ataque do Dolev Yao vamos usar como exemplo um protocolo de troca de mensagens criptografadas simples. Abaixo temos os passos de troca de mensagens entre dois agentes: Ana (A) e Beto (B):

1. $A \rightarrow B : \{M\}_{K_B}$

2. $B \rightarrow A : \{M\}_{K_A}$

Primeiro Ana usa a chave pública do Beto (K_B) para criptografar a mensagem M. Beto recebe a mensagem e por ser o único que possui a chave privada correspondente, pode verificar a mensagem M original. No segundo passo Beto faz o inverso, ele envia uma mensagem M para Ana, criptografando a mensagem com a chave pública (K_A). Ana, por ser a única detentora da chave privada correspondente pode então ler a mensagem.

Mas o protocolo é inseguro e pode ser quebrado, como iremos demonstrar abaixo. Vamos supor que haja na rede um interceptador C, Carlos, que pode ouvir, capturar e modificar toda e qualquer mensagem trocada entre Ana e Beto.

1. C forja uma mensagem M para B fingindo ser A. Ele pode fazer isso pois a chave de A é pública. Ele só precisa forjar o remetente.

2. C intercepta a mensagem $\{M\}_{K_B}$ enviada originalmente por A. Isso é feito para que B não saiba qual o verdadeiro endereço de A.

3. Como C enviou uma mensagem para B no item 1, B responde para C a seguinte mensagem: $B \rightarrow C : \{M\}_{K_C}$.

4. C recupera essa mensagem M, violando o sigilo da comunicação entre Ana e Beto.

5. Além disso, C pode reencriptar a mensagem da seguinte maneira: $C \rightarrow A\{M\}_{K_A}$. Dessa maneira, A e B nem percebe que existe alguém interceptando a comunicação.

Ao analisar o protocolo acima, é fácil perceber que um dos problemas está na identificação de quem envia a mensagem criptografada. Isso pode ser facilmente contornado, incluindo identificador na mensagem criptografada do agente. Por exemplo, a primeira mensagem ficaria da seguinte maneira: $A \rightarrow B\{M, A\}_{K_A}$. Mas será que isso resolve o problema do protocolo?

Isso vai depender do identificador A escolhido para ser concatenado à mensagem M. É preciso garantir que haja uma autenticação entre as partes antes da troca de mensagem. Ou seja, é preciso que Ana confie que está falando realmente com Beto e vice-versa. Se essa parte de autenticação falhar, não podemos garantir que as partes estão falando realmente com quem pretende e o interceptador Carlos pode assumir a personalidade de qualquer participante. Esse tipo de ataque é chamado *Unknown key share attack* (BERNSTEIN, 1999) e é um grande problema nos protocolos assimétricos.

No aplicativo *Signal* nós temos servidores centrais onde os usuários registram suas chaves e identidades (*ClientID*). Portanto, para Beto saber quais chaves Ana utiliza (e vice-versa), ele consulta um servidor do aplicativo e o servidor envia as chaves correspondentes do usuário. Dessa maneira fica mais difícil para Carlos dizer a Ana que ele

é Beto. Ana vai confiar somente nas chaves enviadas pelo servidor. Mesmo nessa arquitetura, como veremos mais tarde, o protocolo pode estar suscetível ao ataque UKS (BERNSTEIN, 1999).

Há ainda o caso do Whatsapp, que implementa um outro modo de garantir a identidade da outra parte. Ele possui uma sequência numérica para cada par de usuário e que pode ser verificada via QR Code ou pela verificação visual. Na figura 10 temos uma imagem da tela do Whatsapp em que esse código de identificação pode ser comparado com o existente no aplicativo do outro participante da conversa. Esse QR Code pode ser visualizado selecionando a conversa com a pessoa que deseja comparar os códigos e indo na opção de Criptografia.

Figura 10 – QR Code Whatsapp

Fonte: Aplicativo Whatsapp (2019)

3.3 MODELO DE ATAQUE AO PROTOCOLO SIGNAL

Um dos grandes problemas dos protocolos assimétricos é a confiança de que uma determinada chave realmente pertence a pessoa com quem se pretende comunicar. Supondo que Ana nunca tenha falado com Beto, então ela não tem como verificar que a chave que algum agente envia a ela é realmente do Beto. Carlos poderia, por exemplo, estar na rede da Ana, interceptar o pedido e enviar a chave dele no lugar da chave do Beto. Ao mesmo tempo, ele pode se comunicar com Beto, fechando um canal seguro entre eles e também

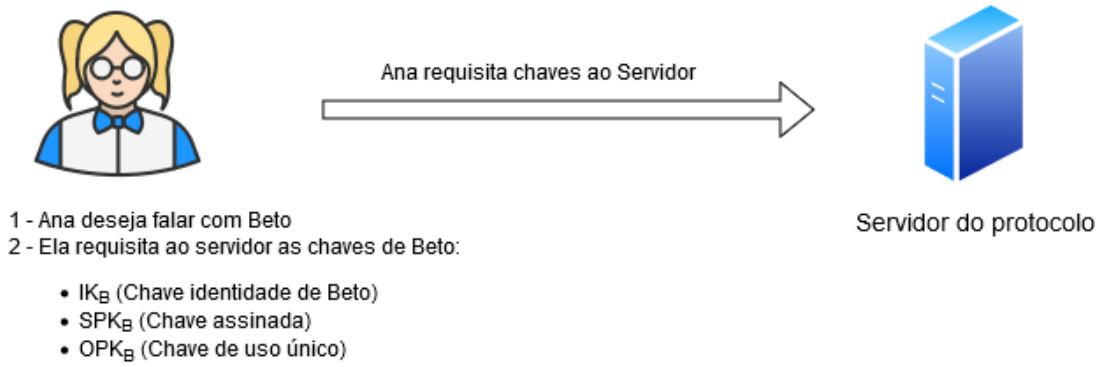
um com Ana. Esse tipo de ataque é conhecido como *Man in the middle* ou simplesmente como MiTM e é bastante comum entre hackers.

Falamos na sessão anterior que o protocolo ponta-a-ponta está sujeito a um ataque chamado *Unknown key share attack*, ou UKS (BERNSTEIN, 1999) . Nesse ataque, o agente se aproveita do desconhecimento da vítima da verdadeira chave da pessoa com quem quer se comunicar pela primeira vez. Por exemplo, no protocolo SSH, considerado o mais seguro para acessar terminais remotamente, usa o modelo *Trust on first use* (TOFU) para conferir a autenticidade do servidor no qual o cliente está se comunicando. Ou seja, no primeiro acesso, o cliente é questionado se quer salvar aquela chave de identidade como sendo a autêntica do servidor. Respondendo "Sim", essa informação é gravada em um arquivo chamado *know_hosts* e servirá com identificador em futuras comunicações com ele.

No protocolo *Signal* existe uma terceira parte que participa da comunicação: o servidor. Os servidores do *Signal* cuidam de guardar todas as chaves (públicas) de identidade de todos os usuários que instalam a aplicação. Então, quando alguém tenta se comunicar com um agente no qual não possui a identificação ainda, ele a solicita a algum servidor do protocolo, como explicado na seção 2.2. Obviamente se Carlos conseguir, por exemplo, simular ser o servidor na rede da Ana ou o servidor do aplicativo for comprometido por algum invasor, toda a comunicação futura fica comprometida.

O ataque iniciaria antes do fechamento da sessão, no protocolo *X3DH*, descrito na seção 2.2. Na figura 11 ilustramos uma requisição de Ana ao servidor. Para iniciar a comunicação com Beto, ela precisa das chaves públicas dele.

Figura 11 – Ana requisita ao servidor as chaves de Beto.

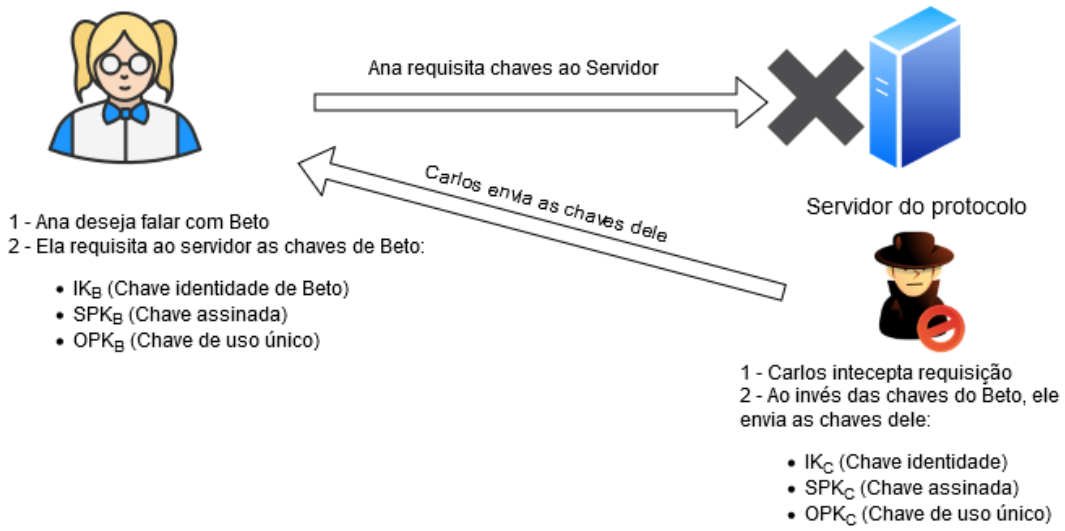


Fonte: Autoria própria

Na figura 12 mostramos como Carlos agiria interceptando a comunicação da Ana e enviando as chaves dele ao invés das chaves do Beto. Em posse das chaves de Carlos, Ana gera a chave compartilhada (*Shared Key* ou *Secret Key*), como mostrado na figura 13 e essa chave é usada por Ana para criptografar a mensagem inicial e enviar para o

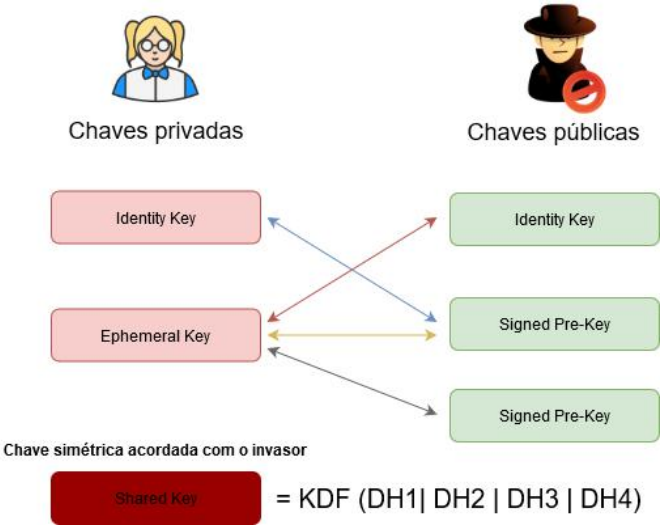
destinatário.

Figura 12 – Carlos envia as chaves dele para Ana.



Fonte: Autoria própria

Figura 13 – Ana forma chave compartilhada com as chaves de Carlos.

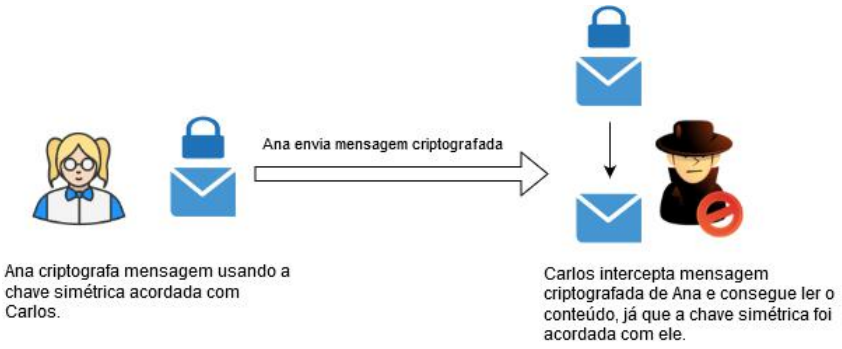


Fonte: Autoria própria

Mas como essa mensagem foi gerado em conjunto com as chaves de Carlos, ele consegue chegar na mesma chave que Ana criou. Portanto, Carlos consegue descriptografar a mensagem e ler seu conteúdo, como mostramos na figura 14.

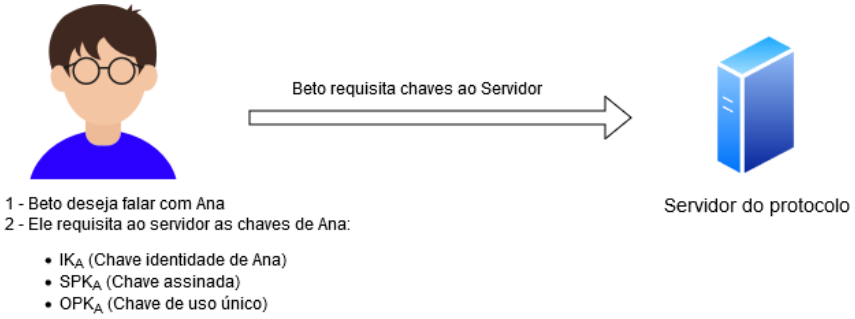
Carlos pode ainda completar o ataque fazendo algo semelhante com o lado do Beto. Na figura 15, Beto solicita ao servidor as chaves de Ana, mas Carlos intercepta essa requisição e envia as chaves dele, como mostrado na figura 16.

Figura 14 – Mensagem interceptada e lida por Carlos.



Fonte: Autoria própria

Figura 15 – Beto requisita chaves de Ana.



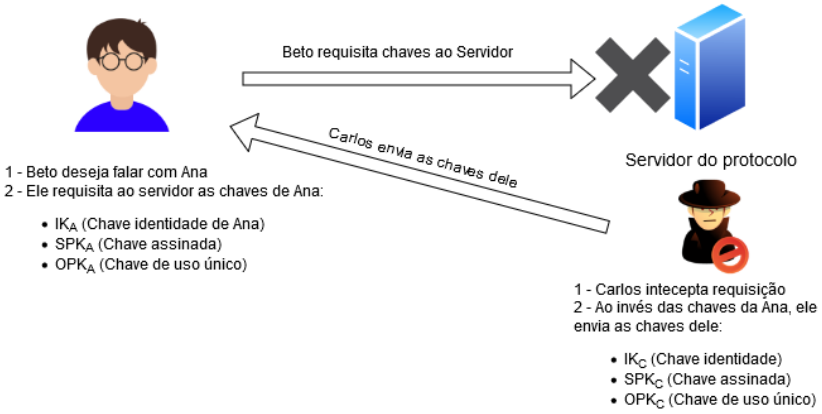
Fonte: Autoria própria

Beto gera sua chave compartilhada com as chaves de Carlos, como ilustrado na figura 17 e, por fim, Carlos consegue ler a mensagem enviada para o Beto, como mostrado na figura 18.

Embora Carlos consiga obter as mensagens trocadas a partir desse momento, ele não conseguiria obter as mensagens anteriores pois além delas não ficarem guardadas no servidor, as chaves utilizadas no momento da criptografia já foram descartadas pelos *endpoints* e já não existem mais. O único lugar que tem as mensagens em texto claro são os próprios *endpoints* de Ana e Beto. Carlos somente conseguirá ler as mensagens a partir do momento em que Ana ou Beto aceitam sua chave com sendo a identidade de cada um. Mesmo nesse caso, se Ana ou Beto já conhecem a chave identidade um do outro (devido a troca de mensagens anteriores), um alerta é mostrado no celular de Ana e Beto, como mostrado na figura 19, no caso de aplicativo Whatsapp.

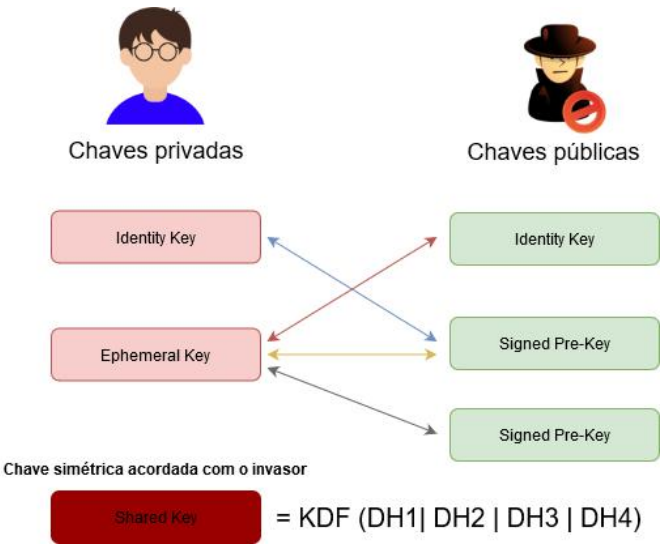
Os servidores do *Signal* servem, portanto, como autoridades certificadoras dos usuários. O *endpoint* do usuário do protocolo guarda informações de todos os usuários que já se comunicaram com ele e o servidor serve também como um reforço para dificultar o ataque.

Figura 16 – Beto requisita chaves de Ana.



Fonte: Autoria própria

Figura 17 – Beto gera chave compartilhada com as chaves de Carlos.



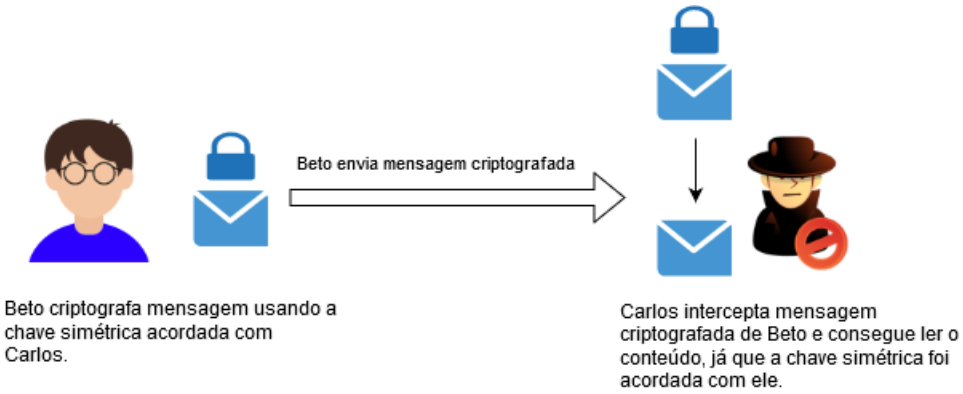
Fonte: Autoria própria

3.4 O SERVIDOR DE CHAVES

Como informado no livro (FERGUSON; SCHNEIER; KOHNO, 2010), o servidor de chaves é uma entidade muito importante em criptografia assimétrica. Basicamente é assumido que todos os participantes compartilham uma chave com o servidor antes de qualquer comunicação com qualquer outro usuário. Por exemplo, Ana configura uma chave K_{AS} que é somente conhecida por ela e pelo servidor. Beto faz o mesmo e configura uma chave K_{BS} entre ele e o servidor.

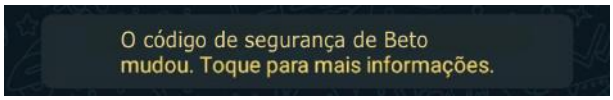
Se Ana quiser se comunicar com Beto e ela não possui nenhuma chave dele, ela confia nessa conexão segura com o servidor para requisitar a chave dele. Como Beto já possui uma comunicação segura com o servidor, ele pode enviar suas chaves para Ana sem se

Figura 18 – Beto requisita chaves de Ana.



Fonte: Autoria própria

Figura 19 – Mensagem exibida quando as chaves de identidade de um usuário mudam



Fonte: Aplicativo Whatsapp (2020)

preocupar com interceptações no meio do caminho. O ponto ruim desse tipo de comunicação é que o servidor serviria como um grande encaminhador de mensagens, tendo o *overhead* de lidar com encriptações e decriptações de vários usuários simultâneos. E, além disso, o servidor conseguiria ler todas as informações trocadas entre Ana e Beto. Porém, isso é facilmente contornado, uma solução melhor seria forçar os participantes trocarem uma chave em comum K_{AB}, por exemplo. E toda comunicação seria realizada com essa chave. E no *Signal* ocorre algo bem parecido, como explicado no Capitulo 2. Exceto pelo fato de o agente que gera as chaves não ser o servidor, mas sim as partes finais, ou seja, Ana e Beto.

3.5 KERBEROS

Não somente o *Signal* usa esse sistema de confiança no servidor e criação de chaves entre os participantes. O Kerberos (NEUMAN; TS'O, 1994), um protocolo amplamente utilizado e que foi baseado no protocolo Needham-Schroeder (FERGUSON; SCHNEIER; KOHNO, 2010), também utiliza algo semelhante, mas com uma principal diferença: no *Signal* o servidor desconhece a chave acordada entre as partes, mas no Kerberos o servidor tem acesso a essa chave.

Basicamente o Kerberos funciona da seguinte maneira: quando Ana quer falar com Beto, ela primeiro contacta o servidor de chaves. O servidor envia uma uma chave K_{AB} e a chave K_{AB} encriptada com a chave de Beto, K_B. Ambas informações são criptografadas

com a chave de Ana, K_A. Dessa maneira somente Ana vai conseguir ler as informações contidas nas mensagens. Ana então encaminha para Beto a mensagem encriptada com a chave dele K_B. Essa mensagem é chamada de ticket. Por ter sido encriptado com a chave de Beto, ele consegue recuperar K_{AB}, que será a chave de sessão, pois só é conhecida por Ana, Beto e o servidor.

O ponto positivo do Kerberos é que ele resolve o problema de processamento no servidor de chaves. A chave de sessão não precisa ser trocada com alta frequência e nem o tráfego precisa passar por ele sempre. O servidor de chaves só precisa manter todas as chaves de sessão trocadas entre os participantes (tickets) pois Ana pode decidir usar um antiga, ou criar uma nova a qualquer momento.

3.6 PUBLIC KEY INFRASTRUCTURE

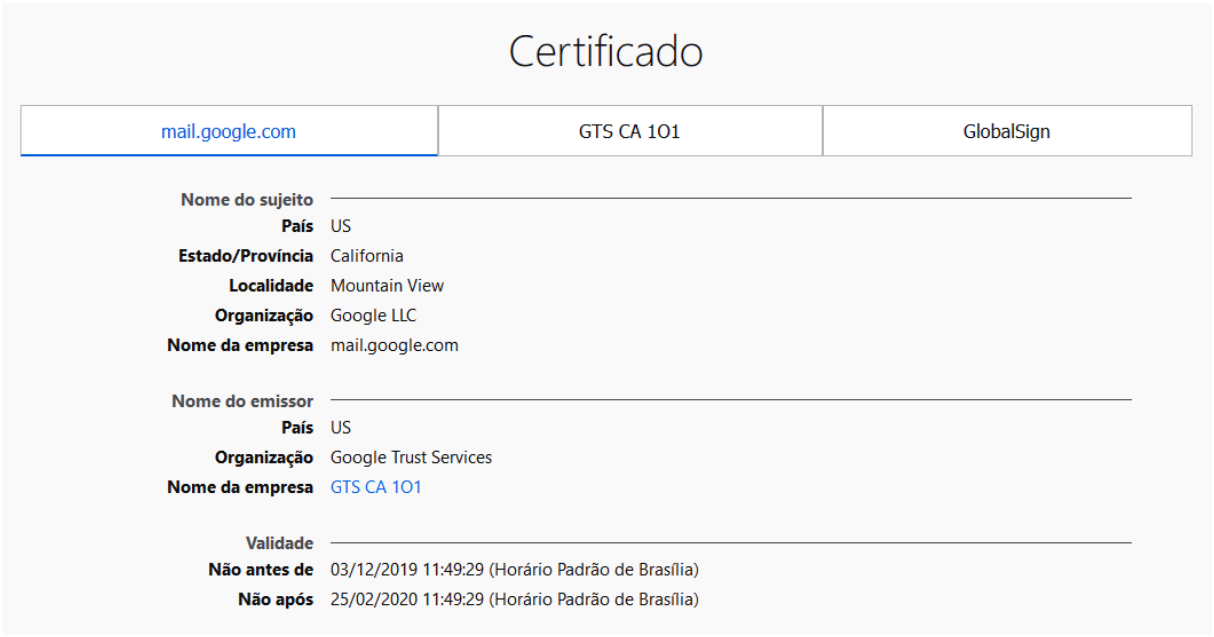
Também conhecido como simplesmente PKI, essa infraestrutura é responsável por guardar todas as chaves públicas das entidades que utilizam. Além de guardar essas chaves públicas, ela é responsável por autenticar se uma chave pertence realmente a pessoa com quem pretendemos comunicar. Por exemplo, Ana pode gerar um par de chaves, pública-privada e enviar a chave pública para a PKI. A chave privada de Ana fica com ela em segredo. Se Beto quiser se comunicar com Ana, ele vai requisitar a chave pública para a PKI, que vai servir como repositório e ponto central de todas as chaves de todos os participantes do protocolo. Beto faz o mesmo, gera um par de chaves e também envia sua chave pública para a PKI. Se Ana quiser falar com ele, requisitará a chave pública dele e criptografará as mensagens com a chave recebida.

A entidade que certifica as chaves dos usuários é chamada de *Certificate Authority* (CA). Ela basicamente é o repositório de chaves. A CA recebe a chave da Ana, verifica a autenticidade dela e gera um certificado atestando que aquela chave pública é realmente da Ana. A CA, sendo uma entidade que participa da PKI, também possui um certificado próprio. Esse certificado atesta a sua autenticidade perante a todos os outros participantes. Por exemplo, ele certifica que o usuário está realmente falando com a Autoridade Certificadora.

Assim como Ana e Beto, a CA gera um par de chaves, chamadas de *Root keys*, e depois gera um certificado atestando sua própria identidade. Esse certificado é especial e tem o nome de Certificado Auto-Assinado. Ele é certificado raiz da PKI.

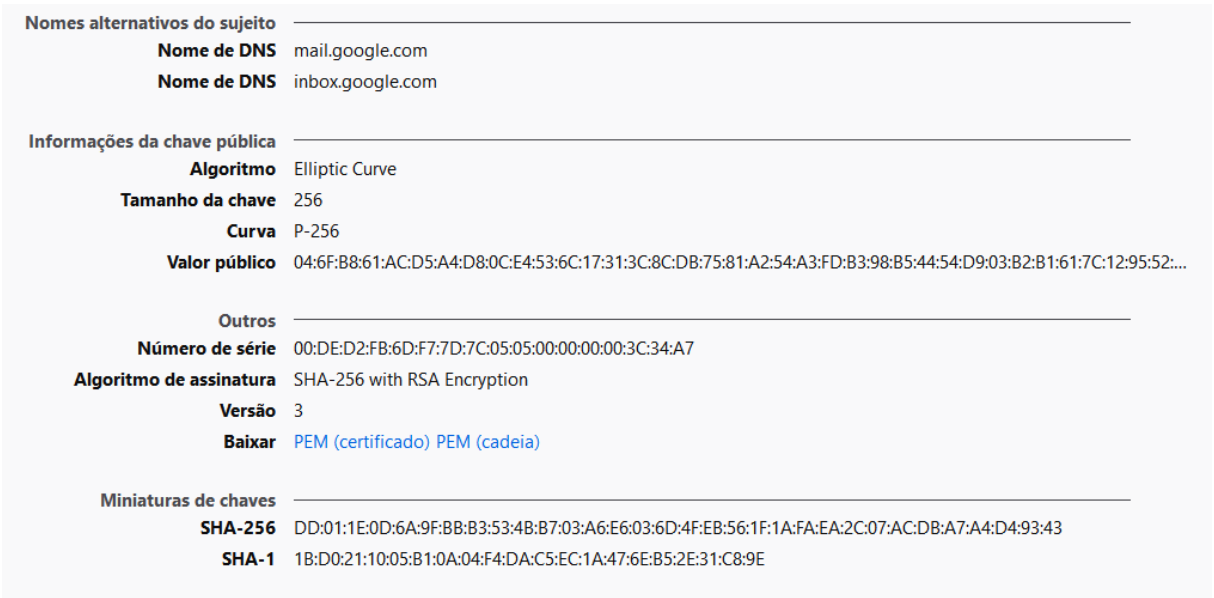
Como podemos notar, qualquer um poderia pegar uma chave pública e gerar um certificado atestando a própria identidade da chave. Mas, o objetivo desse certificado não é garantir a segurança da chave em si, mas sim associar algumas informações de contato a chave, como: dados de contato, data de expiração, lista de permissões etc. Na Figura 20 podemos ver alguns dados de contato para o site mail.google.com. E na Figura 21 temos informações das chaves do mesmo site.

Figura 20 – Exemplo de Certificado de um site



Fonte: Print do Google Chrome (2020)

Figura 21 – Chaves de certificado



Fonte: Print do Google Chrome (2020)

4 CAPÍTULO 4 - MODELANDO O COMPORTAMENTO DO SIGNAL

Neste capítulo vamos modelar o comportamento de algumas rodadas de troca de chaves e mensagens do protocolo *Signal*. Para isso utilizaremos uma linguagem chamada mCRL2(GROOTE et al., 2007) . O mCRL2 foi desenvolvido pelo grupo de análise formal de sistemas na Universidade de Tecnologia de Eindhoven e chefiado por Jan Friso Groote em 2007. É uma linguagem de especificação com a finalidade de analisar, simular e visualizar comportamentos, processos e protocolos.

Os processos são semelhante às funções das linguagens de programação, recebem e enviam dados através de parâmetros. As várias combinações de parâmetros num processo definem um estado específico e um determinado estado pode disparar novas ações. Essas ações por sua vez resultam em novas mudanças de estado. Fazendo um comparativo, podemos representar um programa em mCRL2 como uma máquina de estados em que cada processo tem um espaço de estado correspondente, denominado *Labelled Transitions System* (LTS), que representa todos os estados que um determinado processo pode alcançar, bem como todas as transições para cada estado.

Através de operadores algébricos é possível criar vários processos concorrendo em paralelo. O mCRL2 lineariza todos os processos, removendo assim o paralelismo do sistema. Portanto, sistemas complexos com milhares de processos podem ser traduzidos em um único processo linear. Mesmo sistemas infinitos, como o de troca de mensagens, podem ser linearizados, abstraídos e representados em um espaço de estados finito.

4.1 VISÃO GERAL DO mCRL2

De modo geral, a especificação de um processo no mCRL2 é composta nas seguintes etapas: declaração de ações, definição de processos e na inicialização do sistema.

A declaração de ações inicia-se pela palavra-chave **'act'**. Nela podemos indicar todas as ações do contexto. Além disso, uma ação pode ter vários tipos de objetos, que são separados pelo símbolo '#'.

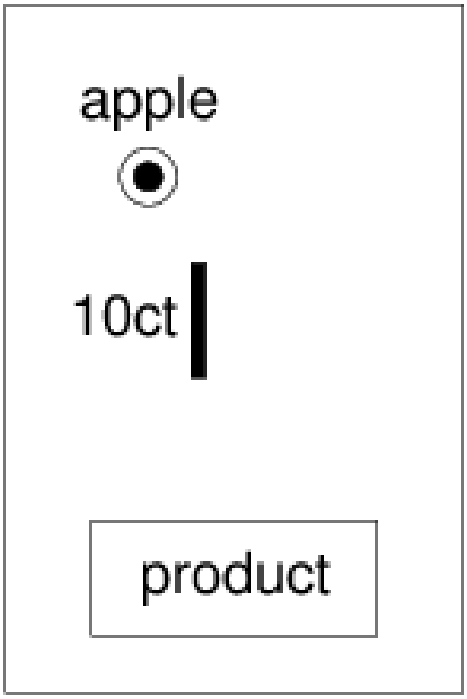
A definição de processos inicia-se pela palavra-chave **'proc'**. Nessa fase são feitas as possíveis combinações de ações que um processo pode realizar. Para ações sequenciais é utilizado o símbolo **'.'**, para ações que podem ocorrer concomitantemente é utilizado o operador '|' e para a escolha entre ações, é utilizado o símbolo '+'. Esses são os principais operadores utilizados na especificação de um processo.

A inicialização de um processo é definida pela palavra-chave **'init'**. Ela pode estar acompanhada somente de um 'proc', que foi anteriormente estabelecido e será executado à partir dele. Os componentes 'allow' e 'comm' são os mais utilizados na sessão do 'init'.

O componente **'allow(A,P)'** remove todos os conjuntos de ações em P que não ocorrem em A. Quaisquer estados que se tornem inalcançáveis também serão removidos pelo mCRL2, pois o sistema resultante é menor. O componente **'comm'** permite que qualquer conjunto de ações sejam alterados para uma única ação.

Na Figura 22 temos uma máquina de vendas primitiva e uma exemplificação simples da especificação em mCRL2 é mostrada na Figura 23.

Figura 22 – Visão geral das ferramentas do mCRL2



Fonte: https://www.mcrl2.org/web/user_manual/tutorial/machine/index.html (2019)

Inicialmente foram declaradas todas as ações que temos no sistema. Há dois processos que foram definidos recursivamente: usuário e máquina.

Neste caso, o usuário é definido com as ações de inserir a moeda de 10 centavos (ins10) e optar pela maçã (optA). A máquina é definida com as ações de aceitar a moeda de 10 centavos (acc10) e colocar a maçã na caixa de produto (putA).

Na etapa de inicialização dos sistema, permitimos, através do componente 'allow' que somente as ações *coin* e *ready* poderão ser visualizadas no sistema.

Além disso, a utilização do componente 'comm' permite a declaração da comunicação do sistema. Definimos os processos usuário e máquina operando paralelamente, de forma que o conjunto de ações ins10 e acc10 sincronizados são transformadas para a ação *coin* enquanto que a sincronização das ações optA e putA são transformadas para a ação *ready*.

O mCRL2 disponibiliza mais de 60 ferramentas para serem utilizadas após a criação das especificações. O uso de tais ferramentas permite ao usuário a análise de sistemas complexos.

Figura 23 – Visão geral das ferramentas do mCRL2

```
act
  ins10, optA, acc10, putA, coin, ready ;
proc
  User = ins10 . optA . User ;
  Mach = acc10 . putA . Mach ;
init
  allow(
    { coin, ready },
    comm(
      { ins10|acc10 -> coin, optA|putA -> ready },
      User || Mach
  ) ) ;
```

Fonte: https://www.mcrl2.org/web/user_manual/tutorial/machine/index.html (2019)

Estão listadas e descritas as principais ferramentas:

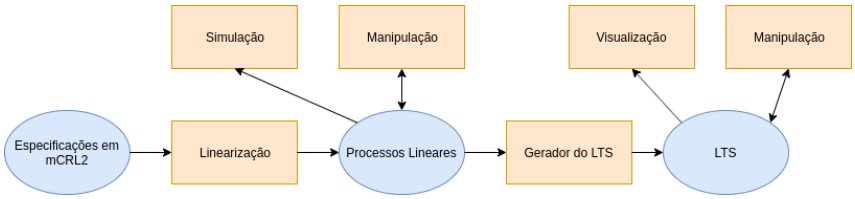
- **mcrl22lps**: Lineariza a especificação mCRL2 e tem como saída o arquivo LPS do tipo binário.
- **lps2lts**: Gera um arquivo de um sistema de transição rotulado (LTS) do tipo binário e tem como saída um arquivo LTS.
- **lps2pbes**: Realiza a leitura de uma fórmula modal e também um processo linear e gera um arquivo de um sistema de equações booleanas parametrizadas (PBES) do tipo binário.
- **pbessolve**: Resolve um arquivo PBES e tem como saída o booleano true ou false.
- **ltsgraph**: A ferramenta desenha gráficos nos formatos 2D/3D.
- **lpsxsim**: Simula indefinidamente a sequência de possíveis ações do protocolo especificado.

Na figura 24 adicionamos um esquema gráfico do que podemos realizar com as ferramentas do mCRL2.

A partir da especificação em mCRL2, os processos são linearizados através da opção mcrl22lps e, do arquivo LPS é possível realizar simulações com uma ferramenta chamada: lpsxsim, que simula o comportamento do sistema para valores randômicos ou sequenciais.

O arquivo LPS fica em formato binário por questões de optimização e não pode ser lido com um editor de texto comum. Se necessário, pode ser lido através da ferramenta

Figura 24 – Visão geral das ferramentas do mCRL2



Fonte: https://www.mcrl2.org/web/user_manual/introduction.html (2019)

lpspp. Da geração do arquivo LPS é possível ainda manipular valores dos processos, como demonstrado na imagem.

Linear Process Specifications (LPS) é uma representação implícita dos espaços de estado dos processos. Do LPS é possível converter para um formato mais explícito, mais descritivo, do comportamento do sistema. Esse novo formato chama-se LTS e pode ser construído à partir do LPS que usa um gerador de espaço de estados.

O LPS gerado no final pode ser aberto com ferramentas gráficas da *switch* de ferramentas do mCRL2. Uma vez aberto, a representação gráfica do LPS nada mais é do que um grafo direcionado. Os estados do grafo representam os estados do sistema e as arestas representam as ações que levam a outros estados. Após a conversão do arquivo para o formato LTS são geradas as visualizações do sistema como um todo.

A verificação da modelagem é realizada usando o *Parameterised Boolean Equation Systems* (PBES). Dado um processo linear e uma fórmula que expressa o comportamento de um processo, um PBES pode ser gerado. A solução do PBES indica se a fórmula criada é verdadeira ou falsa no LPS.

4.2 EXEMPLO DE MODELO DE UM PROTOCOLO SIMPLES EM mCRL2

No exemplo a seguir temos o caso de um protocolo criptográfico no qual é comum termos ações do tipo: envia_mensagem, recebe_mensagem, encoda_msg, decodifica_msg conforme o código mostrado em 4.1:

Código 4.1 – Exemplo de ações em mCRL2

```
act
        envia_msg, recebe_msg, codifica_msg, decodifica_msg ;

proc
        %% Aqui definimos os processos com base nas acoes.

init
        %% Aqui especificamos a inicializacao para rodar simulacoes.
```

Abaixo é mostrado um exemplo de código para o modelo de ataque em um protocolo

criptográfico simples e geral.

Neste exemplo é especificado a troca de mensagens de Ana e Beto sem o interceptador observando a comunicação. É utilizada a notação abaixo para expressar a comunicação dos dois usuários:

1. $A \rightarrow B : \{A, \{M_1\}_{K_B}, B\}$
2. $B : \{IK_B(\{M_1\}_{K_B})\} \rightarrow M_1$
3. $B \rightarrow A : \{B, \{M_2\}_{K_A}, A\}$
4. $A : \{IK_A(M_2)\} \rightarrow M_2$

No passo 1, Ana envia uma mensagem criptografada para Beto usando a chave dele. Beto recebe a mensagem e consegue usar sua chave privada para recuperar a mensagem M_1. Depois, no passo 3, Beto envia uma mensagem criptografada com a chave de Ana e no passo 4 Ana consegue recuperar M_2 usando sua chave privada.

Para ilustrar, no código 4.2 abaixo mostramos um exemplo de código em mCRL2 que pode servir para descrever o protocolo:

Código 4.2 – Exemplo de troca de mensagens entre Ana e Beto

```
sort
        Usuario = struct A | B ;
        Chave = struct Ea | Eb | Da | Db ;
        Mensagem = struct M | Mc ;

act
        encript, decript: Chave # Mensagem ;
        envia_msg, recebe_msg, entrega_msg: Usuario # Mensagem # Usuario
            ;


proc
        UsuarioA = encript(Eb, M).envia_msg(A, Mc, B);
        UsuarioB = recebe_msg(A, Mc, B).decript(Db, Mc).encript(Ea, M).
            envia_msg(B, Mc, A);

init
        allow(
                { encript, decript, entrega_msg},
                comm(
                        {envia_msg | recebe_msg -> entrega_msg },
                        UsuarioA || UsuarioB
                )
        );
```

No trecho **'sort'** do código, definimos três tipos de estrutura: Usuário, Chave e Mensagem. O usuário A é Ana e o usuário B é o Beto. As chaves *Ea* e *Da* representam, respectivamente, chave de encriptação de Ana e chave de decriptação de Ana. Chave de encriptação é a chave pública e a chave de decriptação é a chave privada. Na estrutura de mensagem é utilizada a notação M para definir a mensagem em texto limpo e *Mc* a mensagem criptografada.

No trecho **'act'** são definidas as ações possíveis em nosso modelo. As ações encript e decript aceitam como argumento uma Chave e uma Mensagem. As ações envia_msg, recebe_msg e entrega_msg aceitam dois usuários e uma mensagem.

No trecho **'proc'** são especificadas as sequências de ações permitidas no modelo. No exemplo, o UsuarioA inicia com a encriptação de uma mensagem e a envia para o UsuarioB. Em seguida, o UsuarioB recebe a mensagem encriptada encaminhada do UsuarioA e a decripta com sua chave privada *Db*. Então o UsuarioB encripta uma mensagem de resposta M com a chave pública de A (Ea) e envia a mensagem para o destinatário UsuarioA.

Através de um arquivo mCRL2 é possível visualizar um gráfico LTS à partir da sequência de comandos observados na figura 25

Figura 25 – Comandos

```
$ mcrl22lps arquivo.mcrl2 arquivo.lps

$ lps2lts arquivo.lps arquivo.lts

$ ltsgraph arquivo.lts
```

Fonte: Autoria Própria

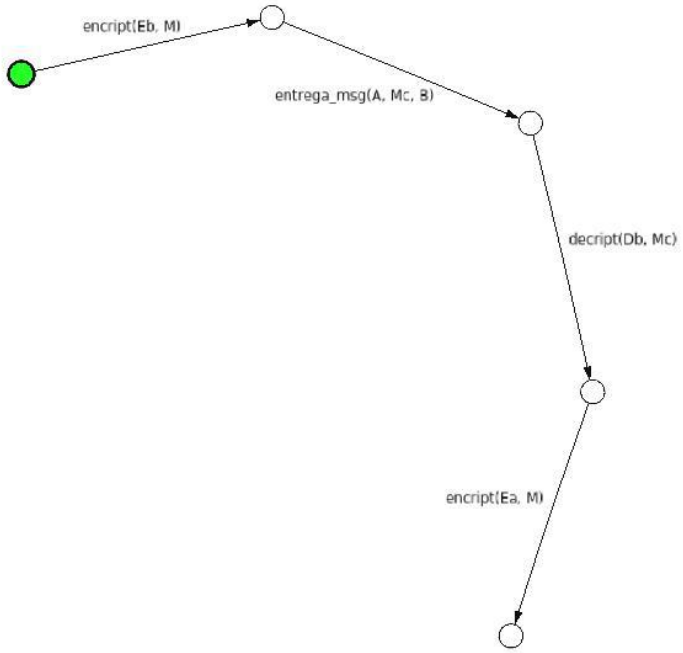
Na figura 26 podemos observar a linearização da modelagem desse protocolo simples, especificado no Código 4.2.

Ao utilizar o modelo especificado no Código 4.2 e adicionando os respectivos identificadores do remetente na mensagem, os passos ficam como mostrado na sequência 1, 2 e 3:

1. $A \rightarrow B : \{A, \{M\}_{K_B}, B\}$
2. $C \rightarrow B : \{C, \{M\}_{K_B}, B\}$
3. $B \rightarrow C : \{B, \{M\}_{K_C}, C\}$

No passo 2, Carlos intercepta a mensagem enviada por Ana e altera o remetente para ele mesmo. Portanto, no passo 3, Beto envia mensagem para Carlos e não para

Figura 26 – Linearização do protocolo simples de troca de mensagens



Fonte: Autoria Própria

Ana, pois o remetente da mensagem anterior estava definido como Carlos. Como Beto codifica a mensagem com a chave de Carlos, ele conseguirá decriptar a mensagem e ler seu conteúdo. Além disso, Carlos pode criptografar novamente a mensagem com a chave de Ana e encaminhar a mensagem original (e até mesmo modificá-la) para ela. Dessa maneira nem Beto e nem Ana conseguem perceber que a mensagem foi interceptada e lida e adulterada por Carlos.

Código 4.3 – Exemplo de interceptação em mCRL2

```
sort
        Usuario = struct A | B | C;
        Chave = struct Ea | Eb | Da | Db | Ec | Dc;
        Mensagem = struct M | McA | McB | McC ;

act
        encript, decript: Chave # Mensagem ;
        envia_msg, recebe_msg, entrega_msg: Usuario # Mensagem # Usuario
            ;

proc

        UsuarioA = encript(Eb, M).envia_msg(A, McB, B) ;
        UsuarioB = recebe_msg(A, McB, B).decript(Db, McB).encript(Ea, M)
            .envia_msg(B, McA, A).UsuarioB + recebe_msg(C,McB,B).decript(
            Db, McB).encript(Ec,M).envia_msg(B,McC,C).UsuarioB ;
        UsuarioC = recebe_msg(A, McB, B).envia_msg(C, McB, B).UsuarioC +
             recebe_msg(B, McC,C).UsuarioC;


init
        allow(
                { encript, decript, entrega_msg},
                comm(
                        {envia_msg | recebe_msg -> entrega_msg },
                        UsuarioA || UsuarioB || UsuarioC
                )
        );
```

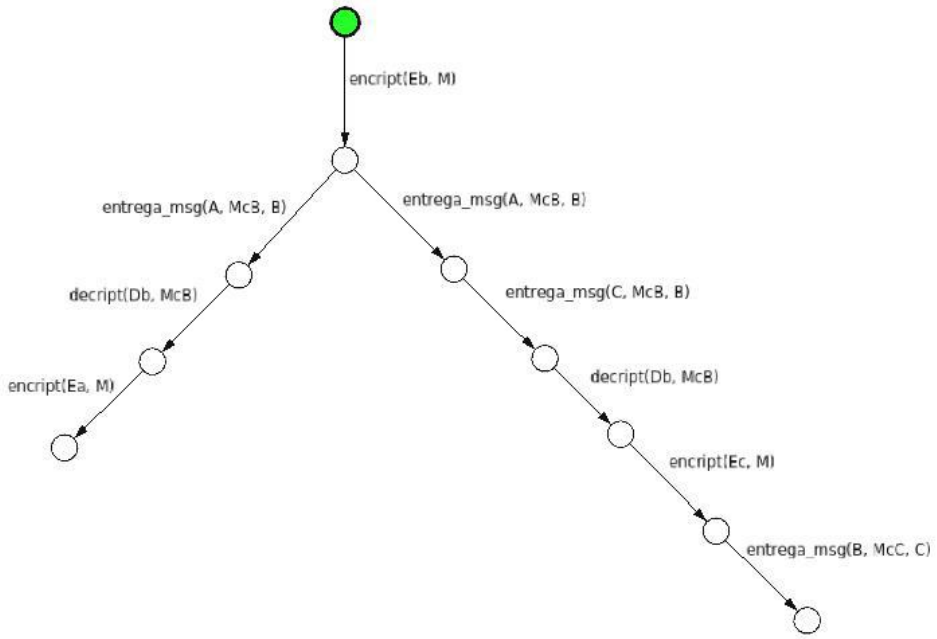
No código em 4.3 há uma possível resolução para esse modelo. Logo será mostrado o grafo depois do modelo linearizado, na figura 27. No grafo, tomamos como ponto de partida o estado em que Ana encriptará uma mensagem para Beto usando uma chave simétrica previamente compartilhada que chamamos de *Eb* no código mCRL2.

4.3 O DOUBLE RATCHET EM mCRL2

Após a explicação acerca da modelagem de um protocolo de troca de mensagens simples, será estendida a mesma lógica para modelar o protocolo Double Ratchet em mCRL2. O Double Ratchet foi explicado em detalhes na seção 2.3 e não será detalhado nessa seção. No código 4.3 foi descrito um trecho simples do início do protocolo.

No código em mCRL2 do Double Ratchet, como simplificação, será convencionado que Ana sempre iniciará a conversa com Beto e, portanto, ela é primeira a solicitar uma chave

Figura 27 – Linearização de interceptação de mensagens entre Ana e Beto



Fonte: Autoria Própria

pública Diffie-Hellman para gerar a sequência de chaves que, posteriormente, gerará uma chave simétrica que servirá para criptografar a primeira mensagem a ser enviada por ela.

Ao receber essa primeira chave pública Diffie-Hellman de Beto, Ana gerará um par de chaves Diffie-Hellman efêmeras e usará a parte da chave privada em conjunto com a chave pública enviada por Beto numa função Diffie-Hellman que terá como saída uma outra chave. Então, essa outra chave será usada como entrada em conjunto com a Root Key da Ana numa função chamada KDF. Como saída teremos uma nova Root Key para Ana e uma Chain Key, que será usada como entrada novamente na função KDF para gerar, finalmente, a Sending Key. Mas, para facilitar a modelagem, será removida essa última passagem pela função KDF que gera a Sending Key. Então, na primeira passagem pela KDF já capturamos essa chave que chamamos de Chain Key e a utilizaremos para criptografar a mensagem para Beto.

No trecho **'sort'** do código foi definida a estrutura **'Usuario'**: A e B para representar Ana e Beto, respectivamente. Também foram adicionados novos tipos de chave na estrutura Chave:

- efe_pubA
- efe_privA
- efe_pubB
- efe_privB

- dhA
- dhB
- rkA
- rkB
- chave_simetrica.

Na tabela 5 foram acrescentadas as definições de cada chave. E na estrutura **'Mensagem'** foram definidos dois tipos de mensagens: a mensagem criptografada, expressa por **'Mcript'** e a mensagem em texto claro, chamada simplesmente de **'M'**.

Tabela 5 – Tabela de chaves do código 4.4

ESTRUTURA	SIGNIFICADO
efe_pubA	Parte pública da chave Diffie-Hellman de Ana
efe_privA	Parte privada da chave Diffie-Hellman de Ana
dhA	Chave gerada por Ana pela função Diffie-Hellman
rkA	Chave Root Key de Ana, gerada em passos anteriores
chave_simetrica	Chave simétrica gerada no final do processo

Fonte: https://signal.org/docs/specifications/doubleratchet (2016)

Código 4.4 – Exemplo do Double Ratchet

```
sort
        Usuario = struct A | B ;
        Chave = struct efe_pubA | efe_privA | efe_pubB | efe_privB | dhA
            | dhB | rkA | rkB | chave_simetrica ;
        Mensagem = struct M | Mc ;

act
        encript, decript: Chave # Mensagem ;
        geraDH: Usuario # Chave # Chave ;
        gera_chave_simetrica: Usuario # Chave # Chave ;
        envia_chave, recebe_chave: Usuario # Chave ;
        envia_msg, recebe_msg, entrega_msg: Usuario # Mensagem # Usuario
            ;

proc

        processo = ( recebe_chave(B, efe_pubB).geraDH(A, efe_pubB,
           efe_privA).gera_chave_simetrica(A, rkA, dhA).encript(
           chave_simetrica, M).envia_msg(A, Mc, B).recebe_msg(B, Mc, A).
           geraDH(B, efe_pubA, efe_privB).gera_chave_simetrica(B, rkB,
           dhB).decript(chave_simetrica, Mc).processo ) + (recebe_chave(
           A, efe_pubA).geraDH(B, efe_pubA, efe_privB).
           gera_chave_simetrica(B, rkB, dhB).encript(chave_simetrica, M)
           .envia_msg(B, Mc, A).recebe_msg(A, Mc, B).geraDH(A, efe_pubB,
            efe_privA).gera_chave_simetrica(A, rkA, dhA).decript(
           chave_simetrica, Mc).processo);


init
        allow(
                { encript, decript, envia_msg, recebe_msg, geraDH,
                   gera_chave_simetrica, envia_chave, recebe_chave,
                   entrega_msg},
                comm(
                        {envia_msg | recebe_msg -> entrega_msg },
                        processo
                )
        );
```

No trecho **'act'** do código 4.4 foi criado somente a ação **'geraDH'**, que representa a função geradora da chave Diffie-Hellman e a ação **'gera_chave_simetrica'**, que representa a função KDF e terá como saída a 'Chain Key', chave simétrica usada para criptografar ou decriptar uma mensagem.

Em **'proc'** há um processo que define trocas de mensagens entre Ana e Beto seguindo

a sequência de geração de chaves do Double Ratchet.

Na figura 29 é apresentado um grafo da linearização do processo. Os itens em vermelho são executados por Ana e os itens em azul são executados por Beto.

Após a ação decriptar, tanto na execução da Ana quanto do Beto, retornamos ao estado inicial do grafo. Isso faz com que as trocas de chaves e mensagens ocorram indefinidamente. Esse retorno ao estado inicial é importante durante a execução da simulação do protocolo ao utilizar a ferramenta **'lpsxsim'**, conforme a imagem 28. A simulação entra em loop até que é forçada a parada. O **'lpsxsim'** pode simular indefinidamente a troca de chaves e mensagens entre Ana e Beto, sempre com a sequência de passos do Double Ratchet.

No trace mostrado na Figura 28, no passo 1, inicialmente, Ana recebe do Beto a chave pública efêmera dele. No grafo da Figura 29 esse trecho corresponde a transição do estado inicial para o primeiro estado em vermelho. Em posse da chave pública de Beto, Ana gera seu par de chaves, captura a chave privada e, concomitantemente, com a chave pública de Beto, gera a saída Diffie-Hellman. No grafo, estamos no segundo estado, em vermelho, e na simulação estamos no passo 2.

Há variações de estado durante a execução até que a primeira mensagem seja enviada para Beto. Na última transição, tanto do lado de Beto, quanto do lado de Ana, temos uma transição chamada 'gera_chave_simetrica', que é a mesma chave encontrada pelos dois participantes do protocolo. Como foi explicado na seção 2.3 essa é uma característica do protocolo Double Ratchet. Essa chave possibilita a leitura do texto criptografado durante a execução do Beto e da Ana. E pela análise do algoritmo, Carlos não tem como gerá-la, já que as informações que podem ser capturadas por ele são somente as chaves públicas da Ana e do Beto. Não há possibilidades do Carlos obter as chaves privadas da Ana e do Beto pois em nenhum momento elas são enviadas para o outro participante.

4.4 ATAQUE AO INÍCIO DE SESSÃO DO SIGNAL

Uma vez iniciada a sessão entre dois participantes no protocolo *Signal*, há bastante complexidade para o agente malicioso intervir no meio da comunicação e conseguir interceptar mensagens, adicionar chaves ou alterar a comunicação. Mas na seção 2.2 mostramos o passo-a-passo de como a sessão entre dois agentes é iniciada.

Nosso objetivo nesta seção é demonstrar um modelo de ataque nessa parte do protocolo. O *X3DH Key Agreement* descreve os passos da troca de chaves iniciais do protocolo *Signal* e na seção 3.3 foi descrito um tipo de ataque em que um agente malicioso compromete um servidor e substitui as chaves da Ana e do Beto pelas chaves dele. Um esquema gráfico é mostrado na seção 3.3.

Nesse cenário vamos supor que Carlos consiga comprometer os servidores responsáveis por repassarem as chaves públicas aos usuários do protocolo. Se Ana quiser falar com

Figura 28 – Simulação de troca de mensagens entre Ana e Beto

LpsXSim

Trace

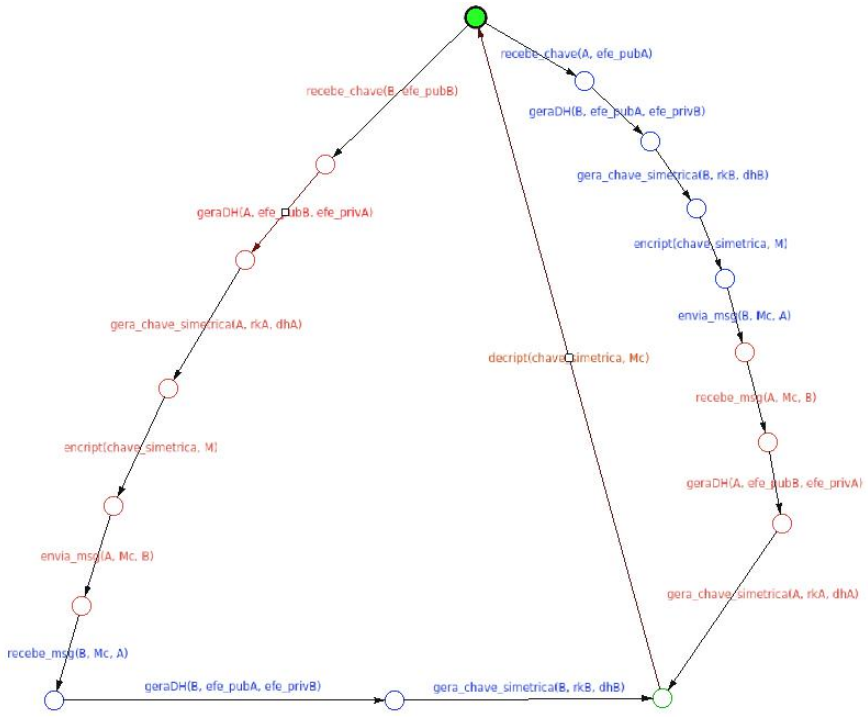
#	Action	State Change
0		s3_processo := 1
1	recebe_chave(B, efe_pubB)	s3_processo := 2
2	geraDH(A, efe_pubB, efe_privA)	s3_processo := 3
3	gera_chave_simetrica(A, rkA, dhA)	s3_processo := 4
4	encript(chave_simetrica, M)	s3_processo := 5
5	envia_msg(A, Mc, B)	s3_processo := 6
6	recebe_msg(B, Mc, A)	s3_processo := 7
7	geraDH(B, efe_pubA, efe_privB)	s3_processo := 8
8	gera_chave_simetrica(B, rkB, dhB)	s3_processo := 9
9	decript(chave_simetrica, Mc)	s3_processo := 1
10	recebe_chave(B, efe_pubB)	s3_processo := 2
11	geraDH(A, efe_pubB, efe_privA)	s3_processo := 3
12	gera_chave_simetrica(A, rkA, dhA)	s3_processo := 4
13	encript(chave_simetrica, M)	s3_processo := 5
14	envia_msg(A, Mc, B)	s3_processo := 6
15	recebe_msg(B, Mc, A)	s3_processo := 7
16	geraDH(B, efe_pubA, efe_privB)	s3_processo := 8

Fonte: Autoria Própria

Beto, ela vai requisitar um conjunto de chaves ao servidor da aplicação, como ilustrado na Figura 1. Se Carlos substituiu as chaves do Beto pelas dele, Ana formará toda a sessão com ele e não com Beto. Se Ana fechar uma sessão com Carlos antes de alcançar o Beto, Carlos terá acesso ao conteúdo de todas as mensagens enviadas por ela. Aqui vale uma observação importante sobre a segurança dos servidores. Não somente Carlos pode comprometer a comunicação entre os participantes, mas o agente que administra os servidores da aplicação também pode formar uma interface intermediária e ficar no meio da comunicação com todos os participantes da aplicação, interceptando toda e qualquer chave trocada entre os participantes. Evidentemente o usuário final pode receber um alerta igual ao mostrado na figura 19, mas esse alerta pode não ter significado relevante, já que sempre que a aplicação é reinstalada há uma mudança das chaves de longo prazo do usuário e a mesma mensagem é exibida. A única maneira de verificar se uma chave pertence mesmo ao usuário final seria comparar manualmente os valores nos celulares dos participantes, como mostrado na figura 10.

Para ilustrar o fechamento de sessão em mCRL2 entre Ana e Beto, definimos a parte de dados no mCRL2 no código 4.5.

Figura 29 – Double Ratchet entre Ana e Beto



Fonte: Autoria Própria

Código 4.5 – Parte dos dados da interceptação das mensagens do X3DH

```
sort
        Usuario = struct A | B | C;
        Chave = struct IKa | IKb | IKc | SPKa | SPKb | SPKc | EKa | EKb
            | EKc | OPKa | OPKb | OPKc ;
        DH1 = struct DH1 ;
        DH2 = struct DH2 ;
        DH3 = struct DH3 ;
        DH4 = struct DH4 ;
        SK = struct SKab | SKac | SKbc ;
        gera_AD = struct ADab | ADac | ADbc ;
```

Primeiro são definidos os usuários que participarão dessa modelagem: Ana (A), Beto (B) e Carlos (C). Depois, definimos as chaves correspondentes aos usuários. Nesse trecho de definição das chaves são utilizadas as nomenclaturas especificadas no Quadro 1. Além disso, são utilizados os dados associados de um usuário (AD), especificado na Seção 2.2. Os trechos que especificam $DH1$, $DH2$, $DH3$, $DH4$ e $gera\_AD$ referem-se, respectivamente, a gera chaves Diffie-Hellman e gera *Associate Data*, explicados na Seção 2.2.

No trecho de código 4.6, são definidas as ações possíveis nas execuções do protocolo e interceptação de mensagem. Essas ações são as de geração de chaves intermediárias, chaves finais ou de dados associados (AD).

As funções que geram chaves intermediárias são as funções DH (Diffie-Hellman) e as chaves finais são as *SecretKeys* (SK). No exemplo, utilizamos todas as quatro partes para gerar a chave final, como especificado na Tabela 3.

A ação enviarNounce do código se refere a primeira mensagem enviada pelo participante. Essa ação usa chave final concomitante com um dado associado (AD).

Código 4.6 – Trecho das ações da interceptação das mensagens do X3DH

```
act
        gera_DH1: Usuario # Chave # Chave ;
        gera_DH2: Usuario # Chave # Chave ;
        gera_DH3: Usuario # Chave # Chave ;
        gera_DH4: Usuario # Chave # Chave ;
        gera_SKa: DH1 # DH2 # DH3 # DH4 ;
        gera_SKb: DH1 # DH2 # DH3 # DH4 ;
        gera_SKc: DH1 # DH2 # DH3 # DH4 ;
        gera_ADa: Chave # Chave ;
        gera_ADb: Chave # Chave ;
        gera_ADc: Chave # Chave ;
        enviarNounce: SK # gera_AD ;
        A_OK, A_NOK, B_NOK, B_OK, ataque_OK, ataque_NOK ;
```

No trecho de código 4.10 dos procedimentos, definimos a ordem de geração de chaves, envio de mensagem e interceptação de Carlos. Como mostrado na Seção 3.1, o modelo Dolev Yao prevê um atacante escutando, interceptando e até mesmo injetando informações no meio da comunicação entre dois ou mais agentes.

Ana inicia a comunicação com Beto. Então é ela quem solicita as chaves Identidade, chave assinada e chave de uso único ao servidor, explicado na Seção 2.2. Mas Carlos intercepta esse pedido e, ao invés de enviar IK_b, SPK_b e OPK_b, ele envia IK_c, SPK_c e OPK_c. Ou seja, Carlos envia suas próprias chaves para Ana. Como é a primeira vez que Ana estará se comunicando com Beto, ela não saberá verificar se as chaves públicas recebidas são de fato as de Beto, então aceitará as chaves de Carlos. Ana gera as quatro chaves Diffie-Hellman e, a partir delas, gera a chave secreta que será usada para criptografar a mensagem inicial, como mostrado na Seção 2.2.

Ana também gera os dados associados usando a chave de Carlos (IK_c) e depois envia um *Nounce*. Carlos vai interceptar essa mensagem e irá conseguir recuperar as informações contidas nela, pois suas chaves públicas foram usadas para encriptar a mensagem. No código em mCRL2 isso é abstraído pela ação 'intercepta'.

Após a interceptação e recuperação da mensagem, Carlos refaz todo o processo que Ana executou, mas agora utiliza suas chaves e as chaves do Beto. Como todo processo é realizado utilizando chaves públicas, Carlos pode facilmente obtê-las. Depois de realizar todo o processo e criar a *Secret Key* entre ele e Beto, Carlos envia um *Nounce* para Beto.

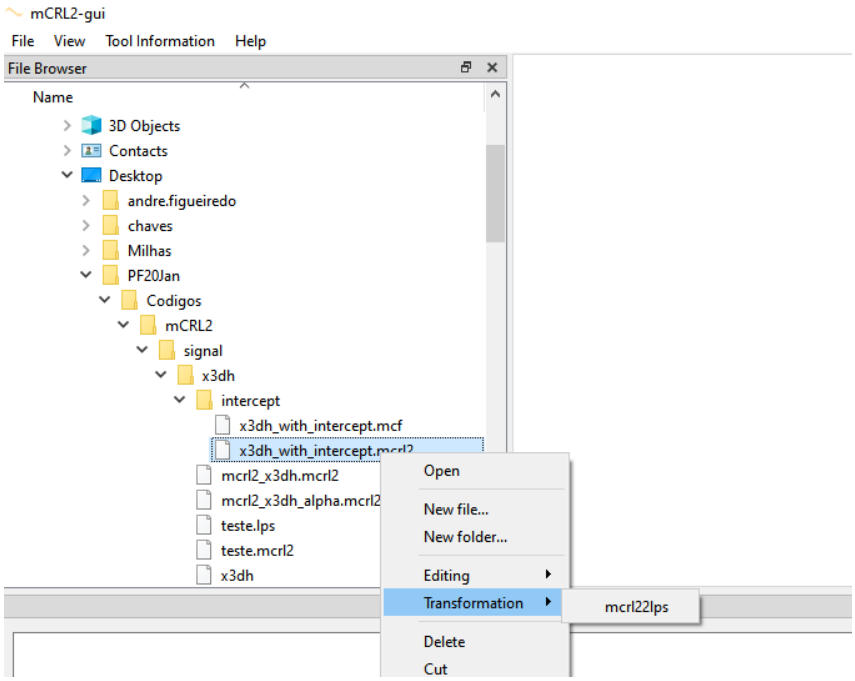
Beto recebe essa mensagem e, como nunca se comunicou com Ana, acredita que as

chaves de Ana são as enviadas por Carlos. Beto refaz todo o processo e responde a Carlos. Pelo mesmo motivo que Carlos conseguiu ler a mensagem de Ana, Carlos também pode ler a mensagem de Beto. Carlos, portanto, fica como intermediário na comunicação entre Ana e Beto. Os dois poderão nunca perceber que existe alguém no meio do caminho que lê as mensagens.

No site oficial do *Signal*, o trecho que é explicado o protocolo X3DH (Seção 2.2), é observado que, para minimizar a eficiência deste ataque, o que pode ser feito é um refinamento das informações contidas nos dados associados (Marlinspike et al., 2016b). Por exemplo, mais informações sobre a identidade dos usuários podem ser adicionadas nos dados associados, de maneira que Ana consiga perceber que não está de fato se comunicando com Beto, mas sim com Carlos. Embora isso seja realmente possível, é necessário ponderar quais informações sensíveis serão adicionadas nesses dados. Uma outra premissa dos aplicativos de troca de mensagens criptografadas é o anonimato e o mínimo de troca de informações pessoais entre os usuários. Portanto os dados associados não podem conter informações muito sensíveis que afetem o anonimato.

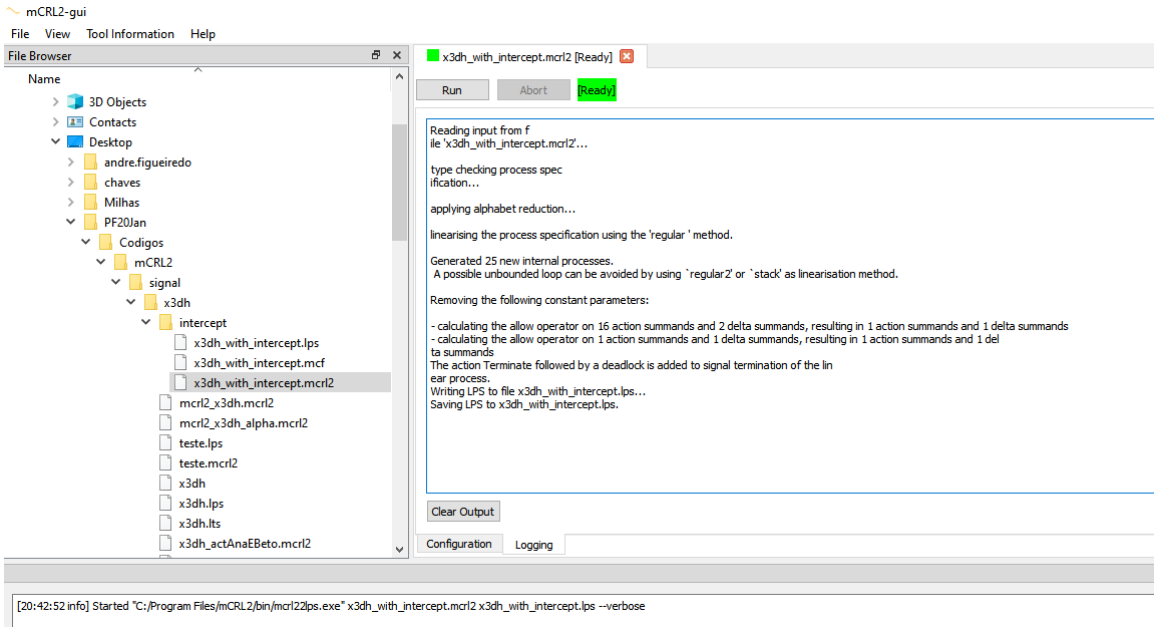
A partir do código de interceptação geramos o arquivo LPS *Linear Process Specification*. Na Figura 30 mostramos como gerar esse arquivo usando a interface gráfica do mCRL2. E na Figura 32 vemos que o arquivo LPS foi gerado com sucesso. Podemos ainda gerar o LTS para visualizar todo o processo em formato de grafo. Como mostrado na Figura 32 isso é feito via interface gráfica do mCRL2 e selecionando a opção lps2lts.

Figura 30 – Gerar LPS



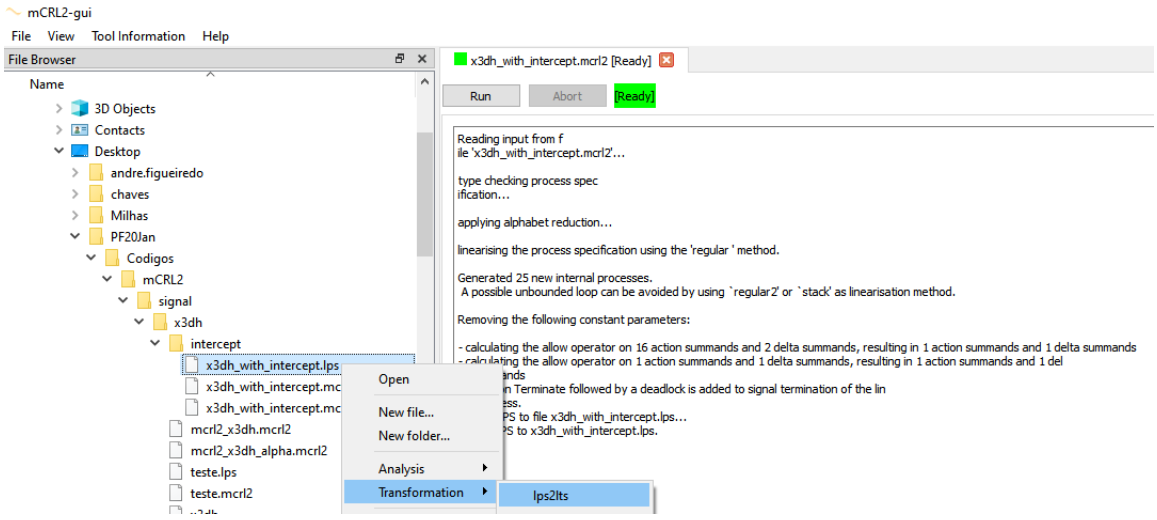
Fonte: Autoria Própria

Figura 31 – LPS gerado com sucesso



Fonte: Autoria Própria

Figura 32 – LPS para LTS



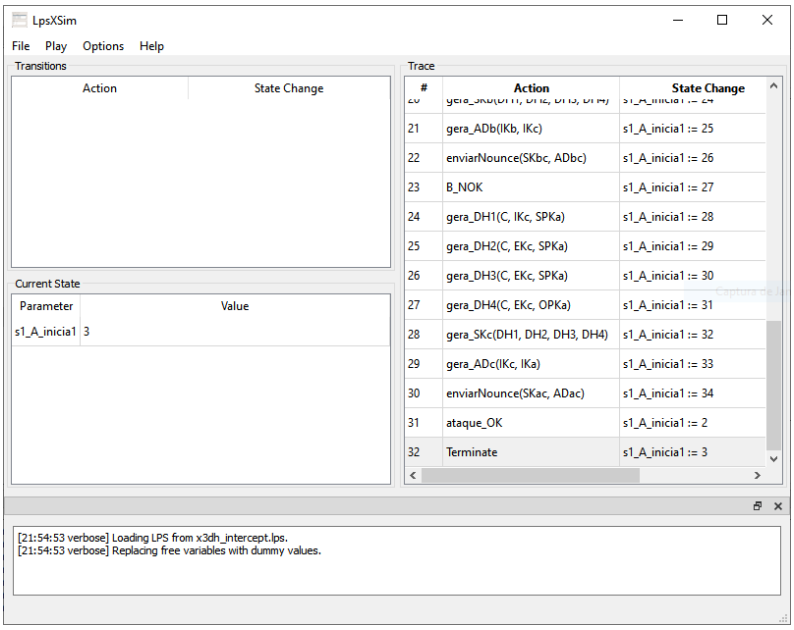
Fonte: Autoria Própria

Podemos simular as transições dos estados do grafo gerado através da ferramenta de interface gráfica do mCRL2 chamada LPSXSIM. Na figura 33 é apresentada a simulação do algoritmo X3DH e na Figura 34 é apresentada a simulação para o algoritmo Double Ratchet.

Outra ferramenta que utilizamos do mCRL2 é o LTSGRAPH, que gera todas as transições de estados realizados no modo gráfico.

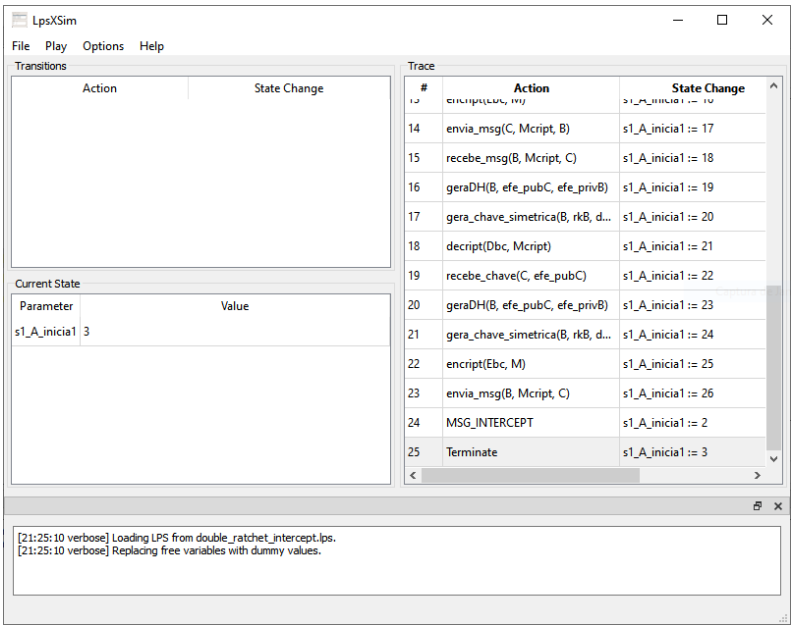
Na Figura 35 temos a representação do algoritmo X3DH, enquanto na Figura 36 é

Figura 33 – Simulação X3DH



Fonte: Autoria Própria

Figura 34 – Simulação Double Ratchet

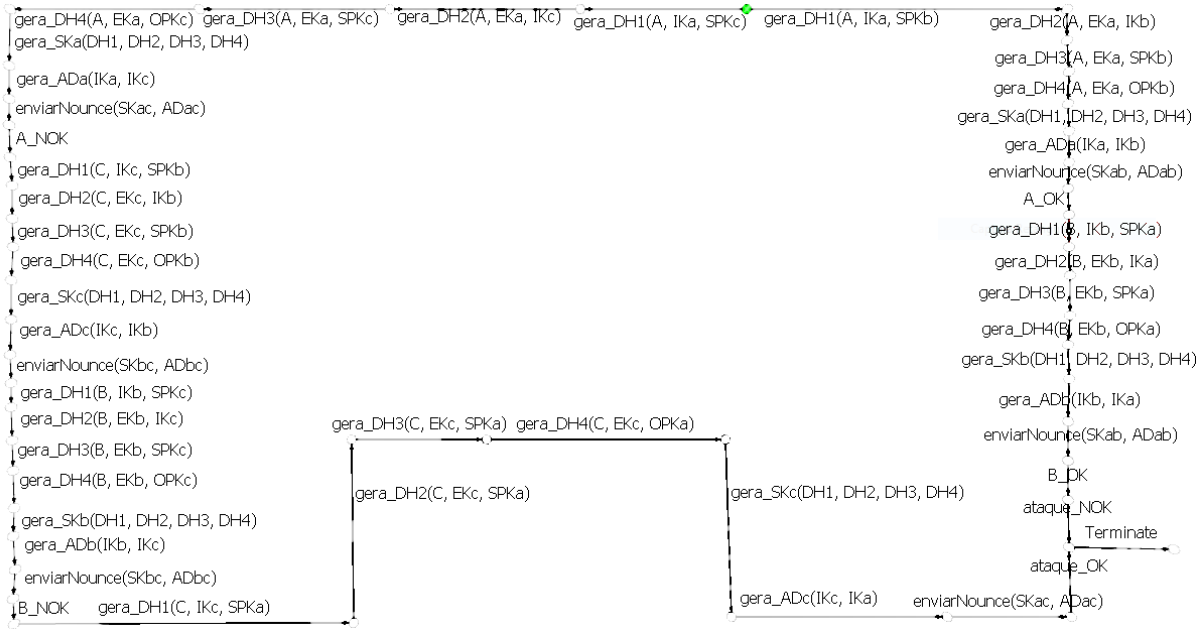


Fonte: Autoria Própria

apresentado o caso do algoritmo Double Ratchet.

Além dos grafos e simulações é possível conjecturar perguntas que tenham como resposta True ou False e testar se um determinado estado é alcançado mediante algumas restrições. Para isso precisamos usar outras duas ferramenta do mCRL2 chamadas: lps2pebs e lps2bool. O lps2pbes, como o próprio nome diz, converte o LPS para um formato PBES

Figura 35 – Gráfico do X3DH



Fonte: Autoria Própria

(*Parameterised Boolean Equation Systems*). Esse formato aceita fórmulas lógicas como parâmetro, permitindo fazer uma verificação do modelo. Como demonstração, usamos duas fórmulas especificadas nos Códigos 4.7 e 4.8 para o caso do protocolo X3DH (Seção 2.2).

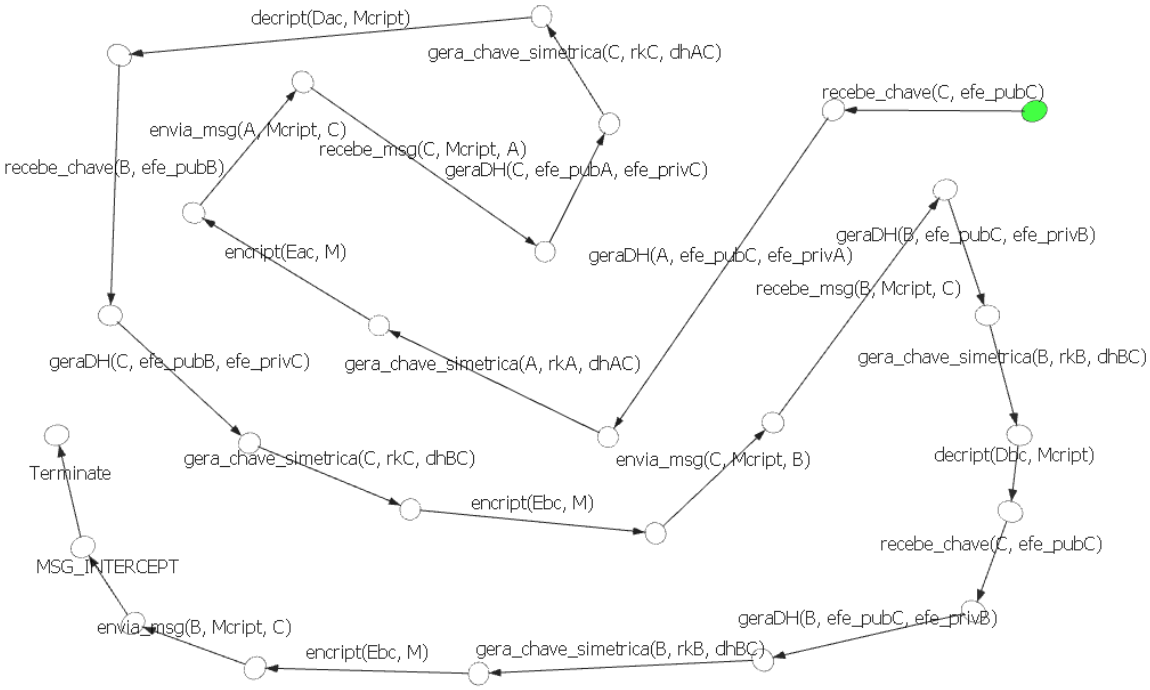
No Código 4.7, '*true**' representa todos as sequências finitas de ações saindo do estado inicial. O ponto '.' representa a transição para um estado que vem após essas sequências de ações. No caso, a chegada ao estado A_NOK, que representa que a participante Ana teve a mensagem interceptada. Logo mais temos uma outra possível sequência de ações seguida de entrada no estado $ataque\_OK$, ação que define que o ataque foi bem-sucedido. Por fim, no último trecho temos uma pergunta: 'true' ou 'false' sobre se aquela série de transições é verdadeira ou não. Se definirmos que ela é verdadeira, como fizemos no exemplo, temos como resposta do programa, 'true', pois chegado o estado A_NOK teremos, por consequência, que houve a interceptação, ou seja, foi atingido o estado $ataque\_OK$. A fórmula citada no código 4.7 representa que após cada A_NOK, seguido de possíveis estados, por fim, somente teremos o $ataque\_OK$.

Código 4.7 – Fórmula PBES para o caso 1

```
[true*.A_NOK.true*.ataque_OK] true
```

No caso 4.8, representa que A_NOK.true.$ataque\_NOK$ é sempre possível. Analisando a fórmula, temos que o mCRL2 retorna 'false', pois para atingirmos o estado $ataque\_NOK$, no qual não houve a interceptação, precisamos que os participantes Ana

Figura 36 – Gráfico do Double Ratchet



Fonte: Autoria Própria

ou Beto não tenham tido suas mensagens capturadas, ou seja, precisamos que ocorra os estados A_OK ou B_OK, por esse motivo o resultado para essa fórmula será 'false'.

Código 4.8 – Fórmula PBES para o caso 2

```
[true*]<A_NOK.true*.ataque_NOK>true
```

Código 4.9 – Trecho dos procedimentos da interceptação das mensagens no X3DH

```
proc
        A_inicia = (gera_DH1(A,IKa,SPKb).gera_DH2(A,EKa,IKb).gera_DH3(A,
           EKa,SPKb).gera_DH4(A,EKa,OPKb).gera_SKa(DH1,DH2,DH3,DH4).
           gera_ADa(IKa, IKb).enviarNounce(SKab,ADab).A_OK.B_respondeA)
           + (gera_DH1(A,IKa,SPKc).gera_DH2(A,EKa,IKc).gera_DH3(A,EKa,
           SPKc).gera_DH4(A,EKa,OPKc).gera_SKa(DH1,DH2,DH3,DH4).gera_ADa
           (IKa, IKc).enviarNounce(SKac,ADac).A_NOK.C_enviaB);

        B_respondeA = gera_DH1(B,IKb,SPKa).gera_DH2(B,EKb,IKa).gera_DH3(
           B,EKb,SPKa).gera_DH4(B,EKb,OPKa).gera_SKb(DH1,DH2,DH3,DH4).
           gera_ADb(IKb,IKa).enviarNounce(SKab,ADab).B_OK.ataque_NOK;

        C_enviaB = gera_DH1(C,IKc,SPKb).gera_DH2(C,EKc,IKb).gera_DH3(C,
           EKc,SPKb).gera_DH4(C,EKc,OPKb).gera_SKc(DH1,DH2,DH3,DH4).
           gera_ADc(IKc,IKb).enviarNounce(SKbc,ADbc).B_enviaC ;

        B_enviaC = gera_DH1(B,IKb,SPKc).gera_DH2(B,EKb,IKc).gera_DH3(B,
           EKb,SPKc).gera_DH4(B,EKb,OPKc).gera_SKb(DH1,DH2,DH3,DH4).
           gera_ADb(IKb,IKc).enviarNounce(SKbc,ADbc).B_NOK.C_enviaA;

        C_enviaA = gera_DH1(C,IKc,SPKa).gera_DH2(C,EKc,SPKa).gera_DH3(C,
           EKc,SPKa).gera_DH4(C,EKc,OPKa).gera_SKc(DH1,DH2,DH3,DH4).
           gera_ADc(IKc,IKa).enviarNounce(SKac,ADac).ataque_OK;


init
        A_inicia;
```

Código 4.10 – Interceptações no Double Ratchet

```
sort
        Usuario = struct A | B | C ;
        Chave = struct efe_pubA | efe_privA | efe_pubB | efe_privB |
           efe_pubC | efe_privC | dhAC | dhBC | rkA | rkB | rkC | Eac |
           Ebc | Dac | Dbc ;
        Mensagem = struct M | Mcript ;

act
        encript, decript: Chave # Mensagem ;
        geraDH: Usuario # Chave # Chave ;
        gera_chave_simetrica: Usuario # Chave # Chave ;
        envia_chave, recebe_chave: Usuario # Chave ;
        envia_msg, recebe_msg, entrega_msg: Usuario # Mensagem # Usuario
            ;
        MSG_INTERCEPT;

proc

        A_inicia = recebe_chave(C, efe_pubC).geraDH(A, efe_pubC,
           efe_privA).gera_chave_simetrica(A, rkA, dhAC).encript(Eac, M)
           .envia_msg(A, Mcript, C).C_recebe;
        C_recebe = recebe_msg(C, Mcript, A).geraDH(C, efe_pubA,
           efe_privC).gera_chave_simetrica(C, rkC, dhAC).decript(Dac,
           Mcript).recebe_chave(B, efe_pubB).geraDH(C, efe_pubB,
           efe_privC).gera_chave_simetrica(C, rkC, dhBC).encript(Ebc, M)
           .envia_msg(C, Mcript, B).B_recebe ;

        B_recebe = recebe_msg(B, Mcript, C).geraDH(B, efe_pubC,
           efe_privB).gera_chave_simetrica(B, rkB, dhBC).decript(Dbc,
           Mcript).B_envia ;

        B_envia = recebe_chave(C, efe_pubC).geraDH(B, efe_pubC,
           efe_privB).gera_chave_simetrica(B, rkB, dhBC).encript(Ebc, M)
           .envia_msg(B, Mcript, C).MSG_INTERCEPT ;


init

        A_inicia;
```

5 CONCLUSÃO

O protocolo *Extended Triple Diffie-Hellman* (X3DH) em conjunto com o *Double Ratchet* são considerados hoje, em 2020, os protocolos mais seguros para troca assíncrona de mensagens, sendo utilizado por grandes empresas como o Facebook (Marlinspike, 2016a) e Google (Marlinspike, 2016b). A Open Whisper System ganhou notória atenção depois de negar acesso a mensagens trocadas entre os utilizadores dos protocolos à autoridades jurídicas e policiais de todo o mundo, incluindo o Brasil. Os serviços do aplicativo chegaram a ser bloqueados no país por algumas vezes como uma espécie de punição da justiça brasileira. E a Open Whisper Systems alega que seu método de criptografia não permite recuperar informações criptografadas trocadas entre qualquer parte que utiliza seus protocolos. Como explicado no Capítulo 2, isso é verídico, visto que não há troca de chaves privadas entre cliente e servidor. O único a conseguir descriptografar as mensagens seriam os pontos finais de comunicação, logo o protocolo está incluído na categoria de criptografia fim-a-fim.

Embora o protocolo tenha essa característica de criptografia fim-a-fim, apresentamos um método na Seção 3.3 em que é possível interceptar no servidor as informações trocadas entre os participantes. Dessa maneira interceptamos as chaves públicas trocadas e as substituímos pela do invasor. Os participantes seriam enganados e achariam que estão se comunicando com quem pretendem. Ilustramos esse cenário usando a linguagem mCRL2 no Capítulo 4 para modelar a sequência de passos que o invasor teria que executar para ter acesso as informações confidenciais.

O mCRL2 permitiu a verificação do protocolo *Signal*, além da visualização do sistema em 2D/3D e a simulação randômica do protocolo. É uma ferramenta promissora para modelagens de sistemas não somente na área de Segurança da Informação, como em diversas outras áreas. Neste trabalho, o protocolo *Signal* foi explorado com algumas restrições e obtivemos resultados positivos em relação a sua vulnerabilidade, ficando em aberto a possibilidade de expansão do protocolo como, por exemplo, a utilização de sistemas paralelos e especificação do algoritmo Diffie-Hellman para obtenção de resultados mais minuciosos.

O ataque utilizado se chama *Unknown key-share attack* (UKS) (BERNSTEIN, 1999) e foi modelado seguindo as premissas apresentadas pelo modelo de ataque Dolev-Yao (DOLEV; YAO, 1983), apresentado na Seção 3.1. Toda a sequência de ações foi modelada usando o mCRL2, que se mostrou uma linguagem poderosa para descrever, validar e verificar sistemas concorrentes.

Embora a linguagem mCRL2 seja bastante rica para modelar sistemas de troca de chaves e mensagens em um sistema criptográfico, ela possui suas limitações. Como mostrado na Seção 4.4 foi necessário descrever o modelo de ataque manualmente para mostrar que o protocolo pode ser comprometido caso o servidor não seja devidamente protegido.

O mCRL2 seria uma linguagens mais descritiva de um modelo de ataque já conhecido. Ela mostraria que um protocolo é inseguro se apresentarmos manualmente um cenário inseguro. A modelagem do protocolo usando a linguagem e suas ferramentas não seriam capazes de dizer se um protocolo é seguro ou não. Seria necessário outras ferramentas mais elaboradas e com esse propósito específico.

O ProVerif (BLANCHET et al., 2018) pode ser usado como complemento para verificar um modelo criptográfico. Em sua página oficial (Blanchet et al., 2016) é possível encontrar exemplos de alguns protocolos já verificados com a ferramenta. A ferramenta usa o Dolev-Yao e cláusulas de Horn para descrever os protocolos. Devido a sua complexidade e envolvimento com trabalhos avançados, modelar o *Signal* usando cláusulas de Horn e ProVerif estão fora do escopo desse trabalho e portanto pode servir de base para trabalhos futuros no campo de lógica e segurança da informação.

A despeito que o *Signal* seja considerado no tempo presente o protocolo público mais seguro para troca de mensagens, ele está sujeito a falhas que somente serão descobertas em pesquisas robustas sobre seu funcionamento. O objetivo desse estudo é analisar o protocolo e apresentar uma linguagem para modelagem da invasão de um agente externo mal intencionado no sistema. A sua licença,disponibilizada em GPLv3 *General Public License* (Marlinspike et al., 2016a), contribui para que qualquer pessoa interessada tenha acesso ao código fonte a fim de explorar conhecimento e incentivar a criação de protocolos com alto nível de confiabilidade e segurança.

REFERÊNCIAS

BERNSTEIN, D. J. Unknown key-share attacks on the station-to-station (sts) protocol. 1999. Acesso em: 20 dez.2019.

BERNSTEIN, D. J. Curve25519: new diffie-hellman speed records. 2006. Acesso em: 19 dez.2019.

BLANCHET, B. et al. Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial. **Version from**, p. 05–16, 2018.

Blanchet et al. **ProVerif: Cryptographic protocol verifier in the formal model**. 2016. <https://prosecco.gforge.inria.fr/personal/bblanche/proverif/>. Acesso em 2020-01-19.

CAMBAZOGLU, V. et al. Modelling and analysing a wsn secure aggregation protocol: A comparison of languages and tool support. Uppsala University, 2015.

COHN-GORDON, K. et al. A formal security analysis of the signal messaging protocol. p. 451–466, 2017.

CONTI, M.; DRAGONI, N.; LESYK, V. A survey of man in the middle attacks. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 3, p. 2027–2051, 2016.

D. Eastlake 3rd, Motorola, P. Jones, Cisco Systems. **US Secure Hash Algorithm 1 (SHA1)**. 2001. <https://tools.ietf.org/html/rfc3174>. Acesso em 2019-12-19.

DOLEV, D.; YAO, A. On the security of public key protocols. **IEEE Transactions on information theory**, IEEE, v. 29, n. 2, p. 198–208, 1983.

FERGUSON, N.; SCHNEIER, B.; KOHNO, T. Cryptography engineering. **Design Princi**, p. 1–30, 2010.

FIPS, F. I. P. S. P. Advanced encryption standard (aes). 2001.

Greenwald et al. **Breach of Ethics**. 2019. <https://theintercept.com/2019/06/09/brazil-lula-operation-car-wash-sergio-moro/>.

GROOTE, J. F. et al. The formal specification language mcrl2. 2007.

HELLMAN, W. D. E. M. E. New directions in cryptography. 1976.

J.CLEMENT. **Number of monthly active WhatsApp users as of 2013-2017**. 2019. <https://www.statista.com/statistics/260819/number-of-monthly-active-whatsapp-users/>. Acesso em 2020-02-06.

Joshua Lund. **Technology preview: Sealed sender for Signal**. 2018. <https://signal.org/blog/sealed-sender/>. Acesso em 2019-12-19.

KRAWCZYK, H. Perfect forward secrecy. **Encyclopedia of Cryptography and Security**, Springer, p. 921–922, 2011.

KRAWCZYK, H.; ERONEN, P. Hmac-based extract-and-expand key derivation function (hkdf). 2010.

Marlinspike. **Facebook Messenger deploys Signal Protocol for end-to-end encryption**. 2016. <https://signal.org/blog/facebook-messenger/>. Acesso em 2020-01-19.

Marlinspike. **Open Whisper Systems partners with Google on end-to-end encryption for Allo**. 2016. <https://signal.org/blog/allo/>. Acesso em 2020-01-19.

Marlinspike et al. **License update**. 2016. <https://signal.org/blog/license-update/>. Acesso em 2020-01-18.

Marlinspike et al. **The X3DH Key Agreement Protocol**. 2016. <https://signal.org/docs/specifications/x3dh/>. Acesso em 2020-01-04.

MARLINSPIKE, M. **Protocolo TextSecure**. 2014. <https://signal.org/blog/the-new-textsecure/>. Acesso em 2019-12-05.

Moxie Marlinspike. **Just Signal**. 2015. <https://signal.org/blog/just-signal/>. Acesso em 2019-12-19.

Moxie Marlinspike. **Signal Foundation**. 2018. <https://signal.org/blog/signal-foundation/>. Acesso em 2019-12-19.

NEUMAN, B. C.; TS'O, T. Kerberos: An authentication service for computer networks. **IEEE Communications magazine**, v. 32, n. 9, p. 33–38, 1994.

PERRIN, M. M. T. The double ratchet algorithm. 2016. Acesso em: 19 dez.2019.

R7, P. **Alerta: 'golpe da festa' usa famosos como isca para invadir celulares e extorquir**. 2020. <https://recordtv.r7.com/domingo-espetacular/videos/alerta-golpe-da-festa-usa-famosos-como-isca-para-invadir-celulares-e-extorquir-19012020>. Acesso em 2020-02-16.

ROGAWAY, P. Authenticated-encryption with associated-data. p. 98–107, 2002.

STEDMAN, K. Y. e. I. G. R. A user study of off-the-record messaging. 2008.

STEINER, M.; TSUDIK, G.; WAIDNER, M. Diffie-hellman key distribution extended to group communication. Citeseer, 1996.