Explorando o Sistema de Criptografia Signal no WhatsApp

Igor Antunes[1], **Luis Antonio Kowada**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

antunes.igor@hotmail.com, luis@ic.uff.br

Abstract. *In 2016, WhatsApp implemented the end-to-end encryption system Signal Protocol to protect all data transmitted in order to prevent any unauthorized third-party access to that information. However, although the cryptographic system is open source, its implementation has been little published by the company. The purpose of this work is to study the implementation of this system to ensure that WhatsApp uses this encryption system and is unable to access the content of the messages from its users. For this, a code injection was used on an Android device to capture messages and keys, and forward it to a third-party system based on Signal Protocol. The results show that the company uses the system properly, without access to the private keys of users and messages content.*

Resumo. *Em 2016, WhatsApp implementou o sistema de criptografia fim-a-fim Signal Protocol para proteger todos os dados transmitidos, de forma a prevenir que acessos terceiros não-autorizados possam obter informações. Entretanto, apesar do sistema de criptografia ser open-source, a sua implementação foi pouco divulgada pela empresa. O propósito deste trabalho é estudar a implementação deste sistema para garantir que o WhatsApp o utiliza corretamente e que não consegue obter acesso ao conteúdo das mensagens de seus usuários. Para isso, utilizou-se uma injeção de código em um dispositivo Android para capturar mensagens e chaves e encaminhar estas para um sistema terceiro baseado no Signal Protocol. Os resultados mostram que a empresa utiliza o sistema apropriadamente, sem acesso às chaves privadas dos usuários e ao conteúdo das mensagens.*

1. Introdução

Antes da massiva utilização do *WhatsApp*, o SMS era o principal meio de comunicação de texto e um dos mais utilizados do mundo. Esta preferência se dava por alguns dos principais motivos como: baixo custo dos torpedos, comunicação assíncrona, objetividade das mensagens e consequentemente maior economia dos minutos adquiridos em planos de telefonia móvel. Entretanto, devido à evolução tecnológica dos dispositivos móveis com o lançamento dos smartphones e o fácil acesso à Internet de alta velocidade, a limitação deste recurso de comunicação acabara se tornando cada vez mais evidente. Com isso, começaram a surgir diversas iniciativas de aplicativos de mensagem instantânea para smartphone, semelhantes aos atualmente utilizados em computadores pessoais.

No meio destas iniciativas surgiu o *WhatsApp*, criado por Jan Koum e Brian Acton em 2009, funcionários do *Yahoo* e colegas de trabalho por quase 20 anos [10], onde o

principal diferencial do projeto foi a busca pela simplicidade com a utilização do próprio número de celular do usuário como seu identificador. Com isso, o aplicativo conseguia por meio do acesso da agenda telefônica presente no dispositivo, identificar automaticamente todos os usuários que já tinham o *WhatsApp* e iniciar uma conversa tão simples como o SMS, dando maior poder ao usuário para enviar não apenas texto como também documentos, fotos, vídeos e áudios. Devido a isso e a massificação da Internet móvel, o *WhatsApp* tornou-se um dos aplicativos de mensagem instantânea mais populares do mundo, possuindo hoje mais de 1 bilhão de usuários ativos por mês. Mediante a turbulência causada por vazamentos de utilização indevida de informação pessoal, e procurando fortalecer cada vez mais a missão de privacidade dos dados, a empresa fez uma parceria com a *Whisper Systems* para implementar o sistema de criptografia fim-a-fim chamado *Signal Protocol* de forma a proteger todas as mensagens transmitidas entre duas pessoas ou em um grupo com o fim de evitar qualquer acesso indevido de uma terceira pessoa àquela informação, seja ela hacker, governo ou até a própria empresa [11].

Apesar do código-fonte do *Signal Protocol* ser aberto ao público, a sua implementação no aplicativo do *WhatsApp* não é divulgada detalhadamente. Até o momento, a empresa publicou apenas um *whitepaper* [11], onde comenta algumas características inerentes ao protocolo e passos gerais sobre o seu funcionamento. Por isso, apesar de se poder comprovar a segurança do protocol no detalhamento da *Whisper Systems*, o *WhatsApp* não poderia ser considerado tecnicamente seguro, por falta de auditoria para verificação de alegações da empresa. Logo, o objetivo deste trabalho é explorar a implementação do *Signal Protocol* no *WhatsApp*, utilizando ferramentas de análise de código Android e técnicas de interceptação de comunicação, para por fim desenvolver uma análise de segurança da informação em conversas um-a-um e de grupo, visando garantir que as informações transmitidas estão de fato protegidas e confidenciadas ao(s) destinatário(s) da informação.

2. Signal Protocol

O *Signal Protocol* foi desenvolvido com o objetivo de ser um dos mais simples sistemas de criptografia fim-a-fim para mensageria instantânea capaz de garantir a confidencialidade, integridade e autenticidade das mensagens enviadas de forma escalável, visando a privacidade das conversas e consequentemente das pessoas que o utilizam. Este sistema pode ser dividido em duas etapas: configuração da sessão, que consiste basicamente em estabelecer uma chave secreta compartilhada entre os dois indivíduos a partir de um conjunto de par de chaves pública-privada; e criptografia das mensagens, que consiste em gerar chaves seguras de criptografia simétrica para encriptação do texto a partir da chave secreta compartilhada e par de chaves pública-privada efêmeras. Os algoritmos utilizados em cada etapa são respectivamente o *Extended Triple Diffie-Hellman (X3DH)* e o *Double Ratchet*, ambos principalmente fundamentados na função *Elliptic-Curve Diffie-Hellman (ECDH)* [5] para acordo de chave simétrica a partir de chaves assimétricas utilizando curvas elípticas, e *HMAC-based Key Derivation Function (HKDF)* [4] para derivação de chaves seguras.

2.1. Extended Triple Diffie-Hellman Protocol(X3DH)

O X3DH [7] é um protocolo de acordo de chave secreta com autenticação mútua baseado em criptografia de chave pública e privada, fundamentado nas funções ECDH e HKDF,

com a proposta de garantir a maior segurança e escalabilidade para aplicação de mensageria. Este protocolo define alguns tipos de pares de chaves assimétricas de curvas elípticas que deverão ser gerados por cada indivíduo:

- *Identity Key Pair (IKP)*, que denomina a identidade do usuário e é gerada uma única vez.
- *Signed Pre Key Pair (SPKP)*, que denomina também a identidade do usuário, pois é assinada com a chave acima, entretanto poderá ser renovada ao longo do tempo para garantir a maior segurança.
- Código de assinatura da chave pública da *SPKP* com a chave privada da *IKP*, para validação de integridade das chaves.
- *One Time Key Pair (OTKP) bundle*, pacote de pares efêmeros para a configuração de sessão entre dois usuários.

Com a geração destas chaves, Alice e Bob se autenticam com um servidor intermediário e em seguida são enviadas as chaves públicas de todos os tipos citados, bem como o código de assinatura da *Signed Pre Key*. Com isso, em um cenário onde Alice queira estabelecer uma chave secreta com o Bob, ela deve solicitar a este servidor intermediário a chave-identidade de Bob $I_B^{publica}$, a chave assinada $S_B^{publica}$ e seu identificador S_B^{id}, o código verificador da chave assinada pela identidade $signature(S_B^{publica}, I_B^{privada})$ e uma chave do tipo One Time Key $O_B^{publica}$ com seu identificador O_B^{id}. Caso o servidor não tenha mais chaves do tipo *One Time Key* do Bob, ele se restringe a enviar apenas as anteriores.

Com a posse das chaves, Alice valida a integridade das mesmas de acordo com a assinatura e, caso seja finalizada com sucesso, ela gera primeiramente um par de chaves efêmera E_A e em seguida é calculado um cruzamento de funções ECDH, conforme ilustrado na Figura 1, para por fim gerar a chave secreta compartilhada, a partir de uma derivação de chave HKDF da concatenação dos resultados das funções anteriores:

1. $DH_1 = ECDH(I_A^{privada}, S_B^{publica})$.
2. $DH_2 = ECDH(E_A^{privada}, I_B^{publica})$.
3. $DH_3 = ECDH(E_A^{privada}, S_B^{publica})$.
4. $DH_4 = ECDH(E_A^{privada}, O_B^{publica})$, caso exista $O_B^{publica}$.
5. $SK = HKDF(DH_1||DH_2||DH_3[||DH_4])$.

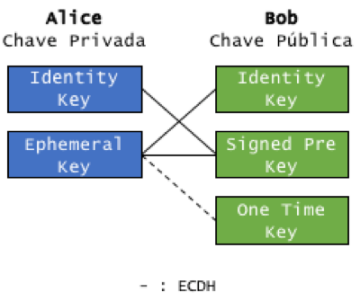

Figura 1. Diagrama do Cruzamento das Funções ECDH

Então, Alice cifra a informação $X = (I_A^{publica}||I_B^{publica})$ com a chave SK utilizando criptografia simétrica AES e envia uma mensagem para Bob com o intermédio do servidor contendo a sua chave pública de identidade $I_A^{publica}$, a chave pública efêmera $E_A^{publica}$,

onde seu par privado foi utilizado nos cálculos acima, o identificador da chave assinada utilizada de Bob S_B^{id}, o identificador da chave do tipo *One Time Key* utilizada de Bob, e por fim uma informação cifrada X, que corresponde às chaves públicas de Alice e Bob encriptada com a *SK*.

Com isso, Bob carrega da sua base a $I_B^{privada}$ e a partir dos identificadores, extrai a $S_B^{privada}$ e $O_B^{privada}$ se houver. Em seguida, Bob calcula o cruzamento das funções de ECDH com os mesmos tipos de pares, invertendo apenas as posições de pública e privada e, com a função de derivação, consegue gerar a mesma chave secreta *SK* calculada por Alice. Entretanto, para validar que funcionou corretamente, Bob decifra a informação *X* e, caso a mensagem decifrada seja a concatenação das chaves públicas de identidade de Alice e Bob, o algoritmo finaliza com sucesso.

2.2. Double Ratchet Algorithm

O algoritmo de *Double Ratchet* [6] é o principal método do *Signal Protocol*, cujo objetivo é gerar chaves efêmeras de criptografia simétrica, para cifrar ou decifrar cada mensagem a partir de uma chave secreta compartilhada e chaves efêmeras que são geradas ao longo da sessão, de forma escalável e com a maior segurança possível. Este algoritmo também é baseado em ECDH e HKDF, sendo que este último é aplicado em um conceito de cadeia de HKDF. A cadeia de HKDF é gerada a partir de uma chave-base e opcionalmente com uma entrada adicional chamada de *salt*, e a saída desta função serve tanto como uma chave de saída a ser utilizada como também de entrada para a próxima função HKDF, fazendo com que estas repetições encadeadas gerem chaves pseudo-aleatórias conforme a necessidade.

Em uma sessão estabelecida entre dois indivíduos, a chave secreta compartilhada é utilizada como entrada do HKDF para gerar a cadeia principal, denominando esta chave como a chave principal. A partir dela, são geradas cadeias de envio e recebimento, e estas cadeias derivadas geram chaves efêmeras de criptografia de mensagem, também chamadas de chave de mensagem, em um algoritmo chamado *Symmetric-Key Ratchet*, que são responsáveis pela decifragem ou cifragem das mensagens.

A derivação da cadeira principal para as chaves de envio e recebimento são realizadas em um algoritmo chamado *Diffie-Hellman Ratchet* e a derivação da cadeia de envio e recebimento para as chaves de mensagem são realizadas em um algoritmo chamado *Symmetric-Key Ratchet*. Esta combinação é apresentada na Figura 2, e recebe o nome de Algoritmo *Double Ratchet*.

2.3. Symmetric-Key Ratchet

Este algoritmo foi desenvolvido basicamente sobre o conceito de cadeia de *HKDF*, onde a chave da cadeia é derivada sucessivamente com a função *HKDF* e a saída de cada função não apenas se torna entrada para a próxima função mas também se torna a chave efêmera de criptografia para uma mensagem específica, chamada de *chave de mensagem*, sendo esta utilizada para cifrar, caso a cadeia seja de envio ou decifrar caso seja de recebimento.

2.4. Diffie-Hellman Ratchet

Este algoritmo foi desenvolvido com o objetivo de renovar as cadeias de envio e recebimento constantemente para evitar que o atacante consiga derivar as chaves de mensagem

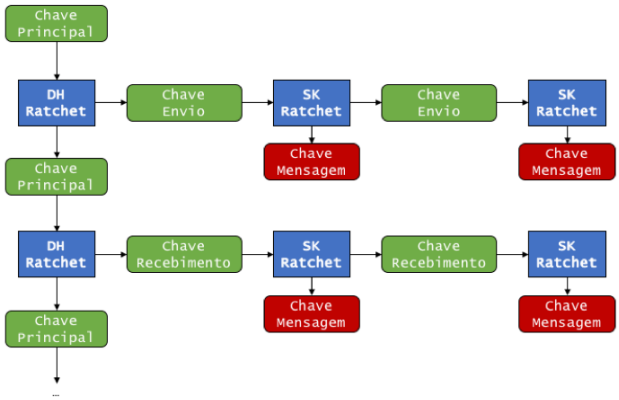

Figura 2. Visão Macro do Algoritmo *Double Ratchet*

indefinidamente e consequentemente todas as mensagens futuras daquela sessão, caso a chave de alguma destas cadeias seja comprometida. A renovação é realizada sempre que o fluxo da mensagem troca de sentido, ou seja, sempre que um usuário que recebeu pela última vez uma mensagem, resolve mandar uma mensagem de volta.

Na inicialização do algoritmo, Alice e Bob geram um par de chaves que se torna o seu *Ratchet Key Pair*, respectivamente R_A e R_B, e Bob envia à Alice a chave $R_B^{publica}$, que em posse desta chave, permite a Alice calcular a chave secreta $SK_1 = ECDH(R_B^{publica}, R_A^{privada})$. Quando Alice envia uma mensagem de volta, ou seja, quando há esta troca de fluxo de mensagem, é enviado à Bob a $R_A^{publica}$, que permite Bob calcular então a $SK_2 = ECDH(R_A^{publica}, R_B^{privada})$ e com isso pode-se assumir que ambos calcularam a mesma chave secreta, pois $SK_1 = SK_2$. Em seguida, Bob gera um novo par de chaves R_B que substitui o anterior para que, assim que Bob enviar uma nova mensagem à Alice, ele envie a nova chave $R_B^{publica}$. Estes dois passos são chamados em conjunto de *DH Ratchet Step*, demonstrado na Figura 3, que é executado sempre que a $R^{publica}$ recebida for diferente da armazenada.

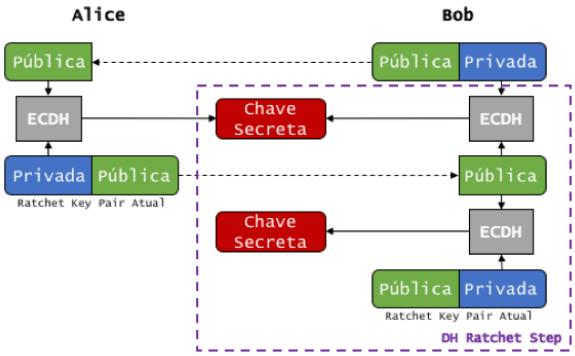

Figura 3. Processo do Diffie-Hellman Ratchet

Logo, as chaves secretas compartilhadas entre as duas partes são as fontes para a geração ou renovação da cadeia de envio para uma parte, e consequentemente recebimento para a outra. A cadeia de envio ou recebimento é gerada a partir da função *Symmetric-Ratchet Key*, onde a chave principal é entrada da função em conjunto com a chave secreta

entrando como *salt*, e a saída desta função não é apenas a chave derivada para a cadeia de envio ou recebimento, como também substitui a chave principal para servir de entrada numa próxima atualização. Este procedimento é ilustrado na Figura 4.

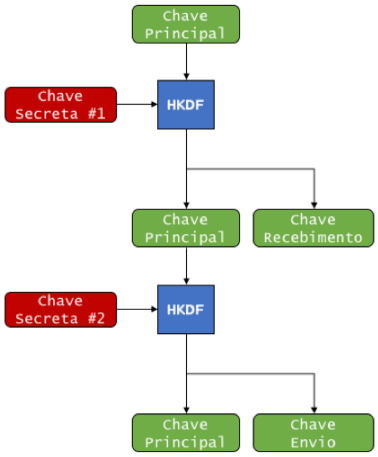

Figura 4. Processo de Derivação da Chave de Cadeia

2.5. Combinação

Considerando que chave da cadeia principal foi gerada por Alice e Bob e a chave de Bob $R_B^{publica}$ foi compartilhada com Alice, ela executa o *DH Ratchet Step* para gerar a chave secreta SK_1, e esta chave secreta renova a chave da cadeia principal e também gera chave da cadeia de envio C_1^E. Com esta chave, é possível executar a *Symmetric-Key Ratchet* para atualizar a chave de cadeia de envio e também gerar a chave de mensagem, para consequentemente enviar uma mensagem cifrada. Caso Alice deseje enviar mais mensagens em sequência, a cadeia de envio continua utilizando a *Symmetric-Key Ratchet* quantas vezes for necessário, até que a troca de fluxo aconteça.

No lado de Bob, cada mensagem enviada de Alice contém em seu cabeçalho $R_A^{publica}$ e na primeira mensagem, quando o Bob avalia que ele não tem a $R_A^{publica}$ armazenada, ele executa o *Diffie-Hellman Ratchet* para gerar a cadeia de recebimento e uma nova R_B para gerar a cadeia de envio de Bob. Com isso, se Bob quiser enviar uma mensagem, utiliza esta cadeia de envio para gerar a chave de mensagem e cifrar. Quando Alice receber a mensagem de Bob, e constatar que a $R_B^{publica}$ não é a mesma armazenada, ela executa o *Diffie-Hellman Ratchet* e o processo, apresentado na Figura 5, se repete indefinidamente enquanto não houver uma sessão válida entre Alice e Bob.

3. Interceptação do Aplicativo

Para a compreensão do funcionamento prático do *WhatsApp*, é necessária a análise do código do aplicativo em alguma plataforma para inspeção da implementação das funcionalidades, e também se monitore a comunicação entre o cliente e o servidor em tempo real para avaliação das trocas de chaves e de mensagens entre duas pessoas ou de um grupo. Na abordagem de análise do código, foi escolhido o ambiente *Android*. Para a análise deste trabalho, foi utilizada a ferramenta FRIDA [2], que faz injeção dinâmica de código, permitindo uma leitura da execução do código analisado em modo de depuração.

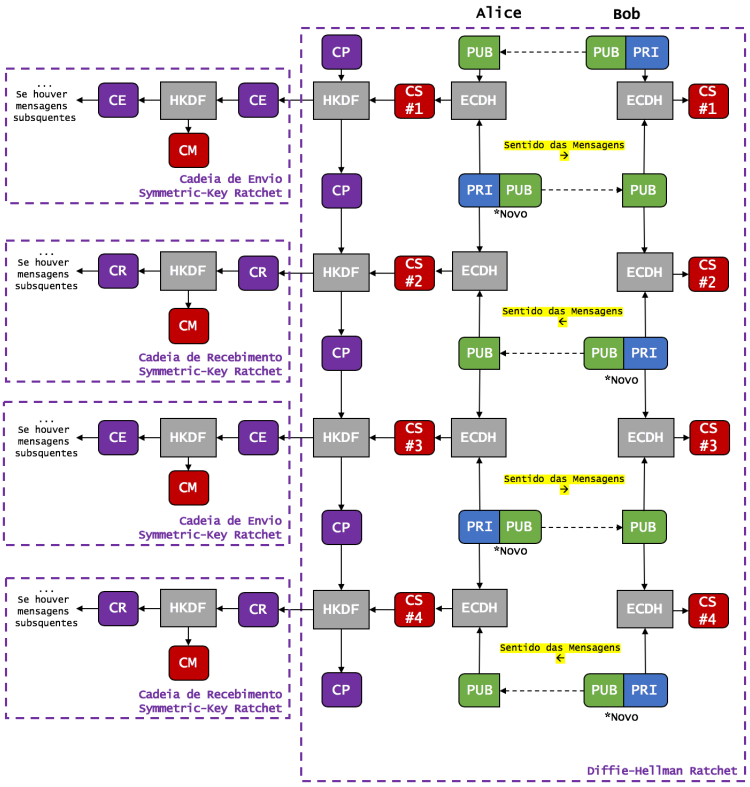

Figura 5. Processo Completo do *Double Ratchet*

Na abordagem de análise de comunicação, foi utilizada a mesma ferramenta para injeção de código nos métodos de escrita e de leitura do canal de comunicação, contornando todas as proteções de criptografia na camada de aplicação e transporte. O código em *JavaScript* a ser escrito é basicamente utilizado para re-implementar algumas funções do aplicativo. Isto é feito primeiro avaliando o código do aplicativo e anotando o método e o caminho completo da classe. Com estas informações, é utilizada a função *Java.use* do FRIDA que captura a classe executada no aplicativo e qualquer objeto instanciado nela, para que seja possível executar o método de implementação.

4. Análise de Código

Para descobrir quais os métodos e as classes avaliados, é necessário primeiramente extrair o código do aplicativo empacotado no formato *APK* para entender melhor o formato da linguagem e a sua estrutura, pois, apesar da linguagem de programação oficial para desenvolvimento de aplicativos ser o Java, a plataforma de execução interpretada adotada é o *Dalvik Virtual Machine*, focando em reduzir ao máximo a utilização de memória e de energia, garantindo a boa concorrência entre múltiplas aplicações. Entretanto, para que a aplicação desenvolvida em Java pudesse ser executada nesta máquina virtual com linguagem de montagem *Dalvik*, eram necessárias que estas aplicações passassem por um segundo processo de compilação, traduzindo os arquivos finais do Java (arquivos de formato CLASS) na linguagem de montagem *Dalvik*, eliminando redundâncias de informação para economia de espaço e integrando-os em um único arquivo de formato *DEX* (*Dalvik EXecutable*).

Como este arquivo *DEX* está em linguagem de montagem *Dalvik*, foi aplicado

um sistema de engenharia reversa que realiza a desmontagem deste arquivo, traduzindo-o em uma linguagem intermediária de mais alto nível, denominado linguagem SMALI, e separando-o em diversos arquivos onde cada um representa uma classe. Visto que o processo de compilação exclui muita informação, não é possível de reverter completamente o arquivo para Java. O código SMALI avaliado do *WhatsApp* contém cerca de 9.400 classes somando mais de 49.000 métodos. Por conta da grande perda de informação gerada no processo de compilação, a maioria das classes e métodos apresentam nomes genéricos muito identificados por letras simples, com exceção das classes primitivas do Java e de algumas classes referenciadas a bibliotecas externas. Por ser um procedimento trabalhoso, a primeira linha de investigação foi realizada em cima de textos estáticos de *logs*. Mas, apesar da classe de *log* estar bem implementada, o mesmo permanece desabilitado por padrão em ambiente de produção e a simples análise dos textos estáticos não contribuiria para o aprendizado do funcionamento do aplicativo, mesmo gerando alguns indícios e referências. Entretanto, com a ferramenta FRIDA, foi possível realizar uma injeção de código nesta classe baseado no Código 1, que permitiu a habilitação de todos os *logs* do sistema, exibindo passivamente as informações que seriam exibidas apenas para os desenvolvedores do aplicativo. Com isso, foi possível identificar os métodos responsáveis pela utilização das chaves do tipo *Identity Key, Signed Pre Key, One Time Key e Ephemeral Key* e consequentemente fazer utilização de injeção de código para capturá-las, conforme descrito no Código 2.

Código 1. Injeção no Método de Informação da classe Log

```
#com.whatsapp.util.Log.info(String information)
Java.use("com.whatsapp.util.Log").i.overload("java.lang.String").
    implementation = function(a){ ... };
```

Código 2. Captura das Chaves

```
#carregamento da identity key pair
Java.use("org.whispersystems.libsignal.d")$init.overload("org.
    whispersystems.libsignal.c","org.whispersystems.libsignal.a.d").
    implementation = function(a,b){ ... };
#carregamento da signed pre key pair
Java.use("org.whispersystems.libsignal.state.h").a.overload().
    implementation = function(){ ... }
#solicitacao da chave one time key baseado em um identificador
Java.use("com.whatsapp.b.f").a.overload("int").implementation =
    function(a){ ... }
#criacao das chaves efemeras
Java.use("a.a.a.a.d").r.overload().implementation = function(){ ... }
```

5. Análise de Comunicação em Tempo Real

Os dois métodos de escrita e leitura da comunicação entre o aplicativo e o servidor foram identificados, dispensando o uso de ataques de *man-in-the-middle* com quebra de protocolo de segurança. Ao avaliar esses métodos, verificou-se que em um recebia como parâmetro *com.whatsapp.protocol.ar* e o outro retornava um *com.whatsapp.protocol.ar*. Avaliando essa classe, foi identificado que se trata de uma estrutura de árvore semelhante ao implementado em XML. Com isso, foi desenvolvida também uma função no FRIDA descrita no Código 3 que, ao injetar o código nos métodos de leitura e escrita, convertesse

esta estrutura em texto e escrevesse na tela. Desta forma, foi viável não apenas analisar todos os *logs* como também entender a comunicação fim-a-fim do aplicativo com o servidor.

Código 3. Captura da Comunicação

```
#metodo de saida de comunicacao
Java.use("com.whatsapp.protocol.u").a.overload("java.io.OutputStream","
    com.whatsapp.protocol.aq").implementation = function(a,b){ ... };
#metodo de entrada de comunicacao
Java.use("com.whatsapp.protocol.t").a.overload("com.whatsapp.protocol.t
    ","java.io.InputStream").implementation = function(a,b){ ... };
```

Desta forma, tornou-se possível a implementação do *Signal Protocol* em um ambiente separado e, conforme o recebimento das chaves do dispositivo e das mensagens, foi possível aplicar o mesmo procedimento e validar a implementação do *Signal Protocol*. Este sistema a parte foi desenvolvido em Java, utilizando Java Web API como interface de comunicação do cliente do FRIDA. Apesar de ter sido importado a biblioteca do *Signal Protocol* para reutilização de código, ele foi pouco utilizado visto que muitas alterações precisaram ser realizadas no meio da implementação do *Signal Protocol* para o correto funcionamento em um ambiente apartado. Ao transmitir as chaves e as mensagens enviadas para este sistema, foi possível reproduzir e decifrar a comunicação tanto enviada quanto recebida pelo usuário deste dispositivo. Como este sistema implementado reproduz fielmente o protocolo, isto prova que de fato o *WhatsApp* implementa de forma correta e consequentemente garante a confidencialidade da informação.

6. Funcionamento do WhatsApp

Com a interceptação do dispositivo, foi possível entender exatamente como o *WhatsApp* funciona no que diz respeito à segurança das mensagens. Nas seções abaixo, será detalhado o processo desde o registro do dispositivo até as conversas propriamente ditas.

6.1. Registro

Quando o aplicativo do *WhatsApp* é instalado pela primeira vez e iniciado a configuração do número de telefone, as seguintes chaves são geradas pelo dispositivo e armazenadas:

- *Registration Id*, identificador do registro do celular randômico
- *SMS Token*, utilizado para gerar a confirmação por SMS
- *Identity Key Pair*
- *Signed Pre Key Pair*
- Código de Assinatura da *Signed Pre Key* pela *Identity Key*
- Pacote de *One Time Keys*

Assim, com o envio das informações de autenticação do dispositivo no servidor, também são enviadas as chaves públicas da *Identity Key Pair* e *Signed Pre Key Pair*, bem como a assinatura para registro no servidor. Após o OK do servidor e a criptografia do canal de comunicação, o dispositivo começa a enviar de forma assíncrona todas as chaves públicas do tipo *One Time Key* para armazenamento no servidor do *WhatsApp*. Neste momento, o dispositivo está pronto para iniciar as conversas seguras com os seus contatos.

6.2. Comunicação

Para a comunicação das mensagens, o *WhatsApp* utiliza uma variação do protocolo *XMPP*, um dos principais padrões *open-source* de mensageria instantânea, com controle de presença, baseado em XML [9]. O *FunXMPP*, desenvolvido pelo *WhatsApp*, tem como principal objetivo reduzir cada vez mais a quantidade de bytes transferidos e consequentemente o consumo de dados dos dispositivos móveis. Como protocolos baseados em *XMPP* são bastante verbosos principalmente nos identificadores das tags, a principal linha de redução do *FunXMPP* foi criar um dicionário das palavras mais utilizadas na comunicação mapeando cada palavra em um byte identificador. Logo, ao invés de enviar a palavra em si, envia-se apenas este byte no local que, com o dicionário atualizado nas duas pontas, ambos vão estar aptos a traduzir o conteúdo, como a Mensagem 4 mostra.

Mensagem 4. Compressão do *FunXMPP*

```
<59 a5="01234567890@91" a7="a2" 44="141765109-2"><12>Teste</12></59>

<message to="01234567890@s.whatsapp.net" type="text" id="141765109-2">
<body>Teste</body></message>

Dic: 59 = message, a5 = to, 91 = s.whatsapp.net, a7 = type, a2 = text,
    44 = id, 12 = body
```

6.3. Conversa 1:1

Antes de Alice enviar uma mensagem para o Bob, ela solicita ao servidor as chaves públicas do Bob para iniciar a sessão, conforme Mensagem 5.

Mensagem 5. Solicitação de Chaves de Bob

```
<iq id=00 xmlns=encrypt type=get to=s.whatsapp.net>
  <key><user jid=bob@s.whatsapp.net></user></key></iq>
```

O servido, em seguida, atende esta solicitação e envia o identificador de registro, a chave pública de identidade, o identificador da chave assinada, juntamente com a sua chave pública e a assinatura, e uma chave pública do pacote de chaves efêmeras de uso único escolhida aleatoriamente junto com o seu identificador, conforme Mensagem 6. O servidor prontamente exclui a chave de uso único enviada, para evitar que a mesma seja enviada novamente para outro usuário.

Mensagem 6. Envio de Chaves de Bob

```
<iq from=s.whatsapp.net type=result id=00>
<list><user jid=bob@s.whatsapp.net><registration>registrationId</
    registration>
<type>05</type><identity>publicIdentitykey</identity>
<skey><id>signedKeyId</id><value>publicSignedKey</value>
<signature>signature</signature></skey>
<key><id>keyId</id><value>publicKeyId</value></key></user></list>
```

Com as chaves em mãos, Alice primeiramente faz a validação da assinatura da chave para garantir a integridade das informações e em seguida utiliza o protocolo X3DH para a geração da chave secreta principal. A partir dela, é gerada a chave da cadeia principal e neste momento a sessão foi estabelecida.Para criptografar a primeira mensagem, Alice

gera a cadeia de envio utilizando a inicialização do *Double Ratchet*, só que utilizando a chave pública do *Signed Pre Key* do Bob utilizado já no protocolo X3DH em conjunto com o seu novo par de chaves efêmero chamado *Ratchet Key Pair*.

Com isso, deriva-se então a chave de mensagem a partir desta cadeia pelo *Double Ratchet* e em seguida utiliza o *AES* para cifrar o texto desejado. Com isso, é gerado então uma estrutura de classe chamada *Signal Message*, que contêm a mensagem cifrada, a chave pública ratchet e um contador que indica quantas derivações da cadeia tem que ser realizadas para chegar na chave de mensagem correta. Posteriormente, é gerado uma estrutura de mensagem chamada *Pre Key Signal Message* que encapsula a *Signal Message* em conjunto com os identificadores e chaves necessárias para geração da chave secreta principal pelo X3DH, e em seguida enviado para o servidor. O servidor então envia para Bob com algumas informações adicionais como o horário da mensagem baseado no servidor e o nome do usuário registrado no aplicativo, conforme descrito na Mensagem 7. Caso Alice queira enviar mais mensagens sem ter recebido o retorno de Bob, ela irá continuar enviando encapsulado em *Pre Key Signal Message* até receber a primeira mensagem do Bob.

Mensagem 7. Envio da *Pre Key Signal Message* para Bob

```
#Alice => Servidor
<message to=bob@s.whatsapp.net type=text id=messageId>
  <enc v=2 type=pkmsg>preKeySignalMessage</enc></message>
#Servidor => Bob
<message   from=bob@s.whatsapp.net
        type=text id=messageId t=timestamp notify=displayName>
  <enc v=2 type=pkmsg>preKeySignalMessage</enc></message>
```

Com o recebimento da mensagem, é inicializado o processo de configuração de sessão, utilizando as chaves públicas de Alice expostas na mensagem e as chaves privadas de Bob baseado nos identificadores contido na mensagem. Em seguida, é realizado o mesmo processo de derivação de cadeia principal para a cadeia de recebimento desta vez, que é igual a cadeia de envio de Alice, e é extraída a chave de mensagem baseado no contador e decifrado a mensagem com AES. No sucesso, é enviado pelo Bob uma mensagem de reconhecimento de entrega ao servidor que repassa a Alice, incluindo o horário, como na Mensagem 8.

Mensagem 8. Reconhecimento da *Pre Key Signal Message*

```
#Bob => Servidor
<ack to=alice@s.whatsapp.net class=message id=msgId></ack>
#Servidor => Alice
<ack to=alice@s.whatsapp.net t=timeStamp class=message id=msgId></ack>
```

Caso Bob queira responder à Alice, primeiramente é necessária a criação da cadeia de envio. Esta cadeia é gerada a partir da chave privada de um novo par de chaves *ratchet* disponibilizado e a chave pública do último *ratchet* da Alice via algoritmo *Double Ratchet*. Assim, com a cadeia gerada, inicia-se o processo de geração da chave de mensagem e, após a cifragem, é criada então a estrutura de *Signal Message* que é em seguida enviada para o servidor, conforme descrito na Mensagem 9. Não é necessário o encapsulamento em *Pre Key Signal Message* pois a sessão já foi configurada e reconhecida pelos dois.

Com isso, o processo se repete de acordo com a troca de sentido do fluxo com *Double Ratchet* e mensagens subsequentes do mesmo fluxo com *Symmetric-Key Ratchet*.

Mensagem 9. Envio da *Signal Message*

```
<message to=alice@s.whatsapp.net type=text id=messageId>
  <enc v=2 type=msg>signalMessage</enc></message>
```

6.4. Conversas em Grupo

O sistema de criptografia das conversas de grupo é diferente de de conversas um-a-um. Nele, define-se primeiramente que cada participante do grupo gera aleatoriamente uma chave de cadeia de envio e também um par de chaves de assinatura. Com isso, cada participante combina a chave de cadeia de envio com a chave pública da assinatura e esta se torna a chave do remetente, também chamada de *Sender Key*, que é utilizada para gerar as chaves de criptografia das mensagens desta pessoa. Por último, cada participante estabelece uma sessão de criptografia fim-a-fim com os outros participantes do grupo, e ao invés de enviar uma mensagem do usuário, envia a sua chave de remetente e, com isso, a sessão de grupo está estabelecida. Se uma pessoa entrar no grupo, cada participante se encarrega de enviar sua chave de remetente via *Signal Protocol* para esse novo participante. Entretanto, sempre que um usuário sai do grupo, o processo de configuração de sessão é reiniciado novamente para garantir que esta pessoa não consiga decifrar as mensagens com as chaves armazenadas dos outros participantes.

Caso um participante queira enviar uma mensagem cifrada para o grupo, primeiramente é derivada a chave de mensagem a partir da chave da cadeia de envio utilizando o protocolo *Symmetric Ratchet Key* e consequentemente atualizando também a chave da cadeia de envio. Então, a mensagem é cifrada com esta chave de mensagem utilizando AES e a mensagem cifrada é assinada com a chave privada de assinatura. Em seguida, esta mensagem e a assinatura são enviadas numa estrutura de classe chamada *Sender Key Message* que é enviada para os participantes por intermédio do servidor. Ao receber esta mensagem, primeiramente o destinatário verifica com a chave pública extraída da chave de remetente se a mensagem de fato é daquela pessoa. Caso positivo, o destinatário realiza o mesmo processo de derivação da chave de mensagem e atualização da cadeia de envio extraída da chave de remetente utilizando o protocolo *Symmetric-Key Ratchet*, e, com isso, decifrar a mensagem utilizando o mesmo protocolo de criptografia simétrica. O processo de cifragem de uma mensagem para o grupo junto com a validação de novos participantes e envio da chave de remetente para estes novos participantes está descrito na Figura 6.

6.4.1. Criação do Grupo e Convite

Quando Alice cria um grupo, é enviada ao servidor uma mensagem de criação com as informações principais deste grupo. Desta forma, quando Alice convida Bob e Carol para participarem do grupo, ela envia uma mensagem para o servidor solicitando a inclusão deles que, por sua vez, verifica o poder administrativo de Alice. Uma vez validado com sucesso, o servidor envia uma mensagem para todos os integrantes do grupo informando a inclusão de Bob e de Alice, conforme formato na Mensagem 10. Isso é necessário para

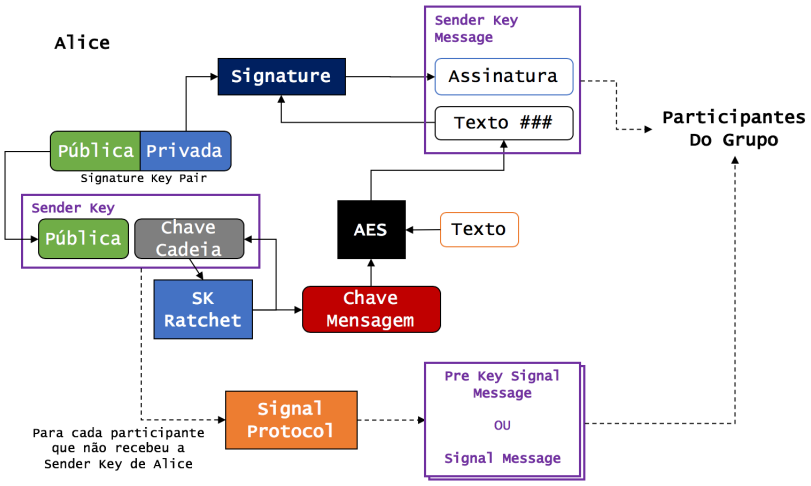

Figura 6. Processo de Cifragem para Grupos

que os outros integrantes possam enviar a sua chave de remetente para o novo participante. Assim, Bob e Carol recebem uma notificação do servidor alertando sobre a inclusão neste novo grupo, como mostra a Mensagem 11.

Mensagem 10. Inclusão de Novos Participantes

```
<notification from=alice-1527446505@g.us type=w:gp2 id=idNotification
   participant=alice@s.whatsapp.net notify=displayName t=timeStamp>
        <add>
                <participant jid=bob@s.whatsapp.net></participant>
                <participant jid=carol@s.whatsapp.net></participant>
        </add></notification>
```

Mensagem 11. Notificação para Novos Participantes

```
<iq from=alice-1527446505@g.us type=result id=a>
        <group id=alice-1527446505 creator=alice@s.whatsapp.net
           creation=timeStamp subject=subject s_t=timestamp s_o=
           alice@s.whatsapp.net>
                <participant jid=alice@s.whatsapp.net type=superadmin>
                </participant>
                <description></description></group></iq>
```

6.4.2. Envio de Mensagem

Antes de enviar uma mensagem para o grupo, Alice verifica se existem usuários para os quais ela não enviou a chave de remetente. Considerando que Bob e Carol ainda não receberam a chave de Alice, esta começa a sessão 1:1 e, no caso de Bob como já foi estabelecida uma sessão com ele, a Alice forma uma mensagem na estrutura de *Signal Message* cifrando a sua chave de remetente. Já no caso da Carol, como é a primeira mensagem, Alice solicita as chaves de Carol e por fim com a execução do *Signal Protocol*, forma uma mensagem na estrutura de *Pre Key Signal Message* com a sua chave de remetente também cifrada. Com isso, Alice cifra a mensagem utilizando a criptografia de

grupo explicada nesta seção, formando a *Sender Key Message* e envia para o grupo como descrito na Mensagem 12. Caso não tenha uma chave de remetente a ser enviada, Alice envia apenas o *Sender Key Message*.

Mensagem 12. *Sender Key Message* com *Signal Message* para Participantes

```
<message to=alice-1527446505@g.us type=text id=idMessage phash=hash>
<enc v=2 type=skmsg>senderKeyMessage</enc>
<participants>
<to jid=bob@s.whatsapp.net><enc v=2 type=msg>SMSG</enc></to>
<to jid=carol@s.whatsapp.net><enc v=2 type=pkmsg>PKMSG</enc></to>
</participants></message>
```

7. Conclusão e Trabalhos Futuros

Este projeto teve por objetivo analisar a implementação do *Signal Protocol* no *WhatsApp* com o fim de garantir que de fato uma criptografia fim-a-fim estava sendo utilizada e consequentemente as mensagens tanto um-a-um quanto de grupo estão confidenciadas aos destinatários. Para verificar isso, foi utilizada a ferramenta FRIDA, para analise do tráfego de troca de chaves e mensagens no meio de comunicação entre o cliente e o servidor e a geração de chaves no dispositivo, de forma a reproduzir os mesmos em um sistema terceiro com o protocolo implementado. Os resultados da análise permitem concluir que, nos dois modos de conversa, o sistema faz bom uso do protocolo de forma robusta e que de fato garante a confidencialidade das mensagens, visto que usam corretamente as funções de ECDH e HKDF e as chaves privadas não são trafegadas.

Para uma análise mais formal da segurança do WhatsApp, pode-se consultar [1, 3, 8].

Com a interceptação da comunicação, existem diversos outros tipos de análise que podem ser realizados. É desejado avaliar o comportamento de outros tipos de mensagem e a possibilidade de manipular as mensagens enviadas e, com isso, avaliar a capacidade do servidor do *WhatsApp* de se proteger de ataques de reprodução, trocando as mensagens enviadas do agente infectado para qualquer destinatário. Além disso, deseja-se explorar futuramente a implementação do *WhatsApp Web* e como é feita a segurança da comunicação do dispositivo móvel com o navegador do computador.

Referências

[1] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 451–466. IEEE, 2017.

[2] FRIDA. Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. `https://www.frida.re`. (Acessado em: 29/06/2018).

[3] Nadim Kobessi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2nd IEEE European Symposium on Security and Privacy*, pages 435–450, 2017.

[4] Hugo Krawczyk and Pasi Eronen. Hmac-based extract-and-expand key derivation function (hkdf). 2010.

[5] Rogerio Albertoni Miranda et al. Criptossistemas baseados em curvas elipticas. 2002.

[6] Signal Org. The Double Ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/. (Acessado em: 29/06/2018).

[7] Signal Org. The X3DH key agreement protocol. https://signal.org/docs/specifications/x3dh/. (Acessado em: 29/06/2018).

[8] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *2nd IEEE European Symposium on Security and Privacy*, pages 415–429, 2018.

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

[10] WhatsApp. Sobre o WhatsApp. https://www.whatsapp.com/about/. (Acessado em: 29/06/2018).

[11] WhatsApp. Whatsapp Encryption Overview. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf. (Acessado em: 29/06/2018).