WHATSAPP WEB REVERSE ENGINEERING

Project report submitted in fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

Puneet Mehta (151365)

Under the supervision of
Dr. Rajinder Sandhu

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"WhatsApp Web Reverse Engineering"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2018 to May 2019 under the supervision of **Dr. Rajinder Sandhu** (Assistant Professor (Senior Grade)-Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Puneet Mehta, 151365

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Rajinder Sandhu
Assistant Professor (Senior Grade)
Department of Computer Science & Engineering and Information Technology
Dated:

Acknowledgment

I would like to express my deep gratitude to the head of Department of Computer Science & Engineering and Information Technology**,** Prof. Dr. Satya Prakash Ghrera for giving me the opportunity to carry out this project. I would like to thank my project guide, Dr. Rajinder Sandhu (Assistant Professor (Senior Grade) - Computer Science & Engineering and Information Technology), for his patient guidance, enthusiastic encouragement and useful critiques of this project.

I would also like to extend my thanks to the lab assistants of the Project Lab of the Computer Science and IT Department, Sh. Ravi Raina and Sh. Sanjeev Kumar for their constant support and help throughout the final year.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

Table of Content

List of Abbreviations

Abbreviation	Explanation
AES	Advanced Encryption Standard
API	Application Program Interface
DH	Diffie-Hellman
ECDH	Elliptic-curve-Diffie-Hellman
E2EE	End-to-End Encryption
GUI	Graphical User Interface
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
HMAC	Hash-based Message Authentication Code
ISP	Internet Service Provider
IV	Initialization Vector
JID	Jabber ID
JSON	JavaScript Object Notation
MAC	Message Authentication Code
QR	Quick Response (code)
REST	Representational State Transfer
SHA	Secure Hash Algorithm
SMP	Socialist Millionaire Protocol
SMS	Short Message Service
URL	Uniform Resource Locator

List of Figures

Abstract

WhatsApp has become a part of the lives of most of the people. It allows people to send messages with the help of the Internet, which means that one no longer have to pay SMS charges (provided the other person also uses WhatsApp). In 2017, there were 1 Billion active users daily and 1.3 Billion active users monthly. Around 55 Billion text messages, 4.5 Billion images and 1 Billion clips/videos were shared per day. Clearly, WhatsApp is here to stay.

Keeping in mind this huge network of people, WhatsApp has recently launched its WhatsApp Business API for medium and large scale businesses. On the other hand, the WhatsApp Business App which can be used by most of the small businesses has very limited functionalities to offer. Talking in numbers, if a company is having more than 500 Employees then only WhatsApp may consider that company for their Business API Access, nothing less than that.

This clearly rules out the option for Individuals, Early Stage Startups, or Small Companies to use that particular platform, leaving a deep hole on the competition side with the big players. So in order to promote equality and fair competition, reverse engineering has been done on the way WhatsApp Web works and code has been written to simulate that. Thereby creating something like the WhatsApp Business API.

Chapter – 1

INTRODUCTION

1.1 Introduction

WhatsApp Messenger is a new age Instant Messenger, having a huge user base of more than 1.5 Billion users sending not less than 60 Billion messages daily [1]. It has literally changed the way, users communicate, share media and has offered a lot of other functionalities. It was completely free a year back, but after Facebook acquired it for a whopping amount of 19 Billion USD Dollars, their revenue model changed completely.

Now the users are still using it for free, not paying anything in real money, BUT there's a glitch. Now, users are paying with their data and details. WhatsApp accepts, that they are using user's chat data to make better-personalized advertisements for them, and unfortunately all of this is Happening after their End-To-End Encryption, which claimed to prevent anyone including WhatsApp from listening to user's conversations.

1.2 Problem Statement

WhatsApp has recently launched its WhatsApp Business API for medium and large scale businesses and the WhatsApp Business App with very limited functionalities for Everyone. Talking in numbers, if a company is having more than 500 Employees then only WhatsApp may consider that company for their Business API Access, nothing less than that. This clearly rules out the option for Individuals, Early Stage Startups, or Small Companies to use that particular platform, leaving a deep hole on the competition side with the big players.

1.3 Objectives

i. To develop a solution that helps Individuals, Early Stage Startups and Small Companies, to use some of the services provided by WhatsApp Business API.
ii. Reverse engineer the way WhatsApp works.
iii. Write a code to simulate the working of WhatsApp. Thereby creating something like the WhatsApp Business API.

Fig. 1.1 Opening gates to better opportunities

1.4 Methodology

In this section, we will discuss the way we carried out the project.

I. Basic Idea:

i. Understand how WhatsApp Web works.
ii. Intercept WhatsApp Web Socket Connection Request & Responses
iii. Write code to simulate the processes taking place in the backend.

II. Tools Used:

i. Charles Web Debugging Proxy: To view web socket connections and incoming and outgoing messages.
ii. Atom: Text editor.

III. Technologies Involved:

i. Web Socket: For communication between browser and server.
ii. Cryptography: WhatsApp communication is based on E2EE.
iii. Reverse Engineering: To understand the way WhatsApp Web works and to implement some of its functionalities.
iv. Cloud Computing: To deploy the final API.

IV. Language Used:

Golang (Go Programming Language)

V. Understanding how things work:

For this task, the communications are read with the help of Charles. Following observations were made while trying to figure out how the things work:

i. On page load (WhatsApp Web), a connection request is sent to the socket.
ii. The web socket is accessible at wss://web.whatsapp.com/ws
iii. This starts the WhatsApp Session.
iv. Requests are sent to the web socket.
v. QR code is generated and displayed on the screen.
vi. Upon scanning the code, a lot of messages are sent in the backend.
vii. Keys are generated (JSON Payload).

VI. Messages:

i. The messages are in plain JSON format.
ii. The E2EE displays data in binary format which can be decrypted using the keys generated while the backend process was taking place. (V.vi.)

Chapter – 2
LITERATURE SURVEY

2.1 Introduction

There is no denial in the fact that technology is deeply integrated into the lives of most of the humans nowadays, and the first word which comes to a layman ’s mind while thinking of technology is ‘Internet’. The Internet has added a whole new dimension to everyone’s lives when it comes to communication, advertisement, entertainment, and the list goes on. This Internet is what has opened doors to new opportunities and innovations. WhatsApp is one such thing. Before going into depth in WhatsApp Reverse Engineering, discussion will be done on WhatsApp in general, in order to understand it in a better way. WhatsApp is an Instant Messenger available for Android, iOS, Web, and Windows [2]. So, it covers pretty much everything. Not only can one send texts, videos, files, audio messages, GIFs, stickers, etc. but it also allows one to make video calls and voice calls. Although one can argue that there are also other platforms such as Facebook, Twitter, Hike, etc. but while talking particularly about Indians, WhatsApp has definitely become a part of their lives. It allows people to send messages with the help of the Internet, which means that one no longer have to pay SMS charges (provided the other person also uses WhatsApp). Over 1 billion people from around 180 countries were using WhatsApp in 2016 every month. In 2017, 1 billion people were using WhatsApp every day [3]. Clearly, WhatsApp is here to stay.

Fig. 2.1. WhatsApp Usage Statistics (2017) [8]

In 2015, WhatsApp introduced WhatsApp Web which allows you to access WhatsApp through the web browser. The conversations are mirrored on the browser from your mobile phone. In order to use WhatsApp Web, all you have to do is, go to https://web.whatsapp.com/ and then you have to follow the given instructions [4]. The main step involved is pairing your phone. In order to do so, you have to scan the QR code with the scanner within WhatsApp. Once the QR code is scanned, you are all set for using WhatsApp Web. The conversations, in this case, are live. You don't have to refresh your phone or web page in order to sync the chats. Logging out from WhatsApp Web is similar to logging out from any other website. You just have to select log out and your session will be terminated. Now, before getting into the technicalities of WhatsApp, how secure WhatsApp is, as claimed by WhatsApp itself.

The main purpose of WhatsApp was to help people communicate with each other. People share their personal moments with the help of WhatsApp. So, in order to maintain security and privacy, WhatsApp has built E2EE in their app [5]. All messages including texts, videos, voice messages and calls - each and everything is secured with the help of E2EE. WhatsApp Platform E2EE is completely dedicated to maintaining security, since, it ensures

people that the messages you are sending can only be read by the person you are communicating with.
Even WhatsApp can't read your messages. Speaking in layman terms, the messages are locked and the only person with the special key to unlock it can read the messages i.e. the receiver/recipient. Even the WhatsApp calls are end-to-end encrypted.

2.2 End-to-End Encryption

WhatsApp E2EE was introduced by WhatsApp in April 2006 in collaboration with Open Whisper Systems. Humans live in the era of digitalization. Most of the things take place over the net. Even communication with each other - personal as well as business, takes place online. So, privacy and security is a major concern. These are the important things which matter to each and every person irrespective of the type of usage. The privacy of a person talking to his/her family member is not less important than that of a person who is talking to a client about the idea of a future product. So, this is where the idea of E2EE comes from. If someone is using your product for the sake of communication, then it is also important to give them the feel of a face-to-face conversation.

For the sake of WhatsApp Reverse Engineering, it is extremely important to understand it's working. So it means that one needs to take a deeper look into the security of WhatsApp.
WhatsApp E2EE, designed by Open Whisper Systems, is based on the Signal Protocol [5]. Think of this protocol as a shield which protects your messages from third party access and also which doesn't allow even WhatsApp to get an access to your conversation in the form of Plaintext. The beauty of this protocol lies in the fact that, even if at a point of time, someone somehow managed to get access to the encryption keys from the user's phone, even then it is not possible to decrypt the messages which were sent previously.

2.3 Signal Protocol: Overview and use in WhatsApp

Before starting up with the technical jargon, a few terms will be discussed to get familiarised with them.

Public Keys

When a message is to be sent, it is first encrypted, and then it is sent. The encrypted message is then decrypted by the recipient. This encryption of the message on the sender's end takes place with the help of Public Key. Private Key is the key corresponding to a particular Public Key which will be used by the receiver/recipient to decrypt the received message. This is what helps in keeping the user's data safe.

Curve25519

Each user has Curve25519 32-byte public key and a 32-byte private key [6]. The sender and receiver will have a 32-byte shared secret which allows the sender to encrypt the messages and then, later on, authenticate them. Computing this shared secret from two public keys is extremely difficult. Curve25519 has extremely high speed, so it can be computed in a very short span of time. Moreover, Curve25519 has very short secret key i.e. 32 bytes only. Curve25519 is an elliptic-curve-Diffie-Hellman function [6] and usually, they have keys of 64 bytes. So that is definitely a plus point of Curve25519. In the case of other functions, the time for key validation is usually quite noticeable but this problem does not arise in case of Curve25519. On comparing it to other functions, one can note that the Curve25519 is twice as fast as them. The best point about Curve25519 is that all the known attacks are quite expensive in nature, which means that Curve25519 is quite secure.

Public Key Types [7]

i. Identity Key Pair: A long-term Curve25519 key pair, generated at install time. The long-term key pair will be stored somewhere deliberately so that it can be used later on at multiple points of time whenever the need arises.

ii. Signed Pre Key – A medium-term Curve25519 key pair, generated at install time, signed by the Identity Key, and rotated on a periodic timed basis.
iii. One-Time Pre Keys – A queue of Curve25519 key pairs for one-time use, generated at install time, and replenished as needed.

Session Key Types [7]

i. Root Key – A 32-byte value that is used to create Chain Keys.
ii. Chain Key – A 32-byte value that is used to create Message Keys.
iii. Message Key – An 80-byte value that is used to encrypt message contents. 32 bytes are used for an AES-256 key, 32 bytes for an HMAC-SHA256 key, and 16 bytes for an IV.

Session Keys are symmetric in nature. These are for one-time use only and are used to encrypting the messages in one chat session.

AES-256 Key

AES (Advanced Encryption Standard) is a popular symmetric encryption algorithm. It performs the calculation on bytes instead of bits. So, when it takes 128 bits of plaintext, it actually treats it as 16 bytes instead. These bytes are then processed as a 4x4 matrix. AES has 3 variants for bit keys - 128, 192 and 256-bit keys. WhatsApp uses the 256-bit keys.

HMAC-SHA256 key

A MAC is a Message Authentication Code, which is used to guarantee that the message while being sent, was not modified. The shared secret key is used to generate and verify MAC. The ‘H’ is used to identify that the MAC is Hash-based. It is constructed from the SHA-256 hash function. In Go programming language, one can use HMAC by importing it with the following statement: import "crypto/hmac"

IV

The IV (Initialization Vector), is a random number which is used along with a secret key while encrypting data. The purpose of the random number is to avoid repetition so that hackers cannot use a dictionary attack by finding some pattern.

2.3.1 Client Registration

When a user/client registers, the public Identity Key, public Signed Pre Key (with its signature), and a batch of public One-Time Pre Keys is transmitted to the server. [7] The public keys are stored by the WhatsApp Server as an identification of the user. However, the private keys are never stored at the WhatsApp Server, in order to maintain the privacy of the user. The private keys are only stored at the user's device.

2.3.2 Initiating Session Setup

In order to start a conversation with another user, first of all, an encrypted session needs to be established. This established session is maintained till an external event does not take place. An external event may include reinstalling the app or changing the device. So, apart from these events, the client doesn't have to rebuild a new session.

Following are the steps involved in building a new session [7]:

i. The person to initiate the conversation or the initiator requests the public Identity Key, public Signed Pre Key and a single public One-Time Pre Key for the person to whom the message is to be sent or the recipient.

ii. The requested public key values are then returned by the server. A One-Time Pre Key, as the name suggests, can be used only. So after being requested, it is removed from server storage.

iii. The recipient's Identity Key, the Signed Pre Key, and the One-Time Pre Key are saved by the Initiator as $I_{recipient}$, $S_{recipient}$ and $O_{recipient}$ respectively.

iv. An ephemeral Curve25519 key pair - $E_{initiator}$, is generated by the initiator.

v. The initiator loads its Identity Key as $I_{initiator}$.

vi. The initiator then calculates a master secret as

$$\text{master\_secret} = \text{ECDH}(I_{initiator}, S_{recipient}) \,||\, \text{ECDH}(E_{initiator}, I_{recipient}) \,||\, \text{ECDH}(E_{initiator}, S_{recipient}) \,||\, \text{ECDH}(E_{initiator}, O_{recipient})$$

Final ECDH is omitted, if there is no one-time pre-key.

vii. The initiator then uses HKDF to create a Root Key and Chain Keys from the master_secret.

2.3.3 Receiving Session Setup

After the long-running encryption session has been created, the sending of messages can be started by the initiator to the recipient. In that case, it doesn't even matter if the recipient is online or offline. All the information that the recipient requires is included by the initiator to build a corresponding session until the recipient responds. The initiator's $I_{initiator}$ and $E_{initiator}$ is included in this. When the message that includes session setup information is received by the recipient [7]:

i. The corresponding master_secret is calculated by the recipient using its own private keys and the public keys which are there in the header of the message sent by the initiator.
ii. The One-Time Pre Key which was used by the initiator is deleted by the recipient.
iii. HKDF is used by the initiator to derive from the master_secret, a corresponding Root Key and Chain Keys.

2.3.4 Exchanging Messages

After the establishment of the session, messages are exchanged by the client which are protected with the help of a Message Key using AES256 for encryption. For authentication, HMAC-SHA256 is used. For each message, the Message Key is unique and ephemeral. This means that the Message Key which was used to encrypt one message cannot be

reconstructed again from that session state after transmission or retrieval of a message. A new Chain Key is created with each message round trip by performing a new ECDH agreement. Forward secrecy is thus provided through the combination of hash and DH ratchet [7].

2.3.5 Calculation of Message Key from Chain Key

Whenever the message sender needs a new Message Key, the calculation is done as follows:

Steps:

i. Message Key = HMAC-SHA256 (Chain Key, 0x01).
ii. Chain Key = HMAC-SHA256 (Chain Key, 0x02).

The Chain Key then "ratchets" forward. Also, the current or past values of Chain Key can't be derived by a stored Message Key [7].

2.3.6 Calculation of Chain Key from Root Key

During each message transmission, the ephemeral Curve25519 is also sent along with it. The new Chain Key and Root Key are calculated after receiving a response. [7] The calculation is as follows:

Steps:

i. ephemeral_secret = ECDH(Ephemeralsender, Ephemeralrecipient).
ii. Chain Key, Root Key = HKDF(Root Key, ephemeral_secret).

A chain is used to send messages only, so as a result of which, message keys cannot be reused. The way of calculation of Message Key and Chain Key makes sure that no problem is faced if the messages are delayed, lost or are out of order.

2.3.7 Transmission of Media as well as Other Attachments

Not only text messages, but messages in the form of images, video, audio or files are sent safely with the help of e2e encryption [7]:

i. An ephemeral 32-byte AES256 and a 32-byte HMAC-SHA256 key are generated by the sender while sending a message.
ii. The attachment is encrypted by the sender with the AES256 key using a random IV in order to avoid dictionary attack. A MAC of the ciphertext is then appended using HMAC-SHA256.
iii. The encrypted attachment is then uploaded by the sender to a blob store.
iv. A normal encrypted message is sent by the sender to the recipient containing the HMAC key, encryption key, the SHA256 hash of encrypted blob, and a pointer to it in the blob store.
v. The message is then decrypted by the receiver. The encrypted blob is retrieved from the blob store and its SHA256 hash is verified. After that, the MAC is verified and the plaintext is decrypted.

2.3.8 Group Messages

In the case of group messages, there are two main strategies, namely, "server-side fan-out" and “client-side fan-out”. In the case of messenger apps without encryption, “server-side fan-out” is employed for group messages. The client transmits a single message. That message is distributed individually to different group members by the server. On the other hand, in the case of “client-side fan-out,” the client itself transmits the message individually to different group members. In WhatsApp groups, the messages are built on the pairwise encrypted sessions to achieve efficient "server-side fan-out" for most of the group messages. The “Sender Keys” component of the SMP is used in order to accomplish the same. A series of events take place the first time a group member wants to send a message:

i. A random 32-byte Chain Key is generated by the sender.
ii. After that, a random Curve25519 Signature Key key-pair is also generated by the sender.

iii. The 32-byte Chain Key and the public key is combined by the sender from the Signature Key into a Sender Key message.
iv. The Sender Key is individually encrypted by the sender to each group member with the help of the pairwise messaging protocol.

The above steps are only followed for the very first message. After that for rest of the messages, the steps followed are [7]:

i. A Message Key is derived by the sender from the Chain Key. After that, the Chain Key is updated.
ii. The message is encrypted by the sender using AES256.
iii. The ciphertext is then signed using the Signature Key.
iv. The single ciphertext message is transmitted by the sender to the server. The server then uses the "server-side-fan-out" approach to send messages. The sender key is cleared when a group member leaves the group.

2.3.9 Transport Security

E2EE is not the only part by which WhatsApp ensures security. While transportation, there is also a separately encrypted channel between WhatsApp client and server. The communication between them takes place completely through this encrypted channel [7]. The client is provided with the following features:

i. The connection setup and resume are extremely lightweight and fast.
ii. Not just messages, but the metadata is also encrypted so that it is hidden from unauthorized network observers. The user's identity thus remains secret.
iii. The server doesn't store any client-related secret. Client Authentication is done using the Curve25519 key pair. So, only the public keys are stored at the server. So, assuming the worst case circumstances, even if someone gets unauthorized access to the server’s user database, the private authentication credentials will still remain safe.

Chapter – 3
SYSTEM DEVELOPMENT

In this chapter, how the various features of WhatsApp are to be implemented will be discussed.

3.1 WhatsApp Web API

WhatsApp Web itself has an interesting API. It can be directly accessed from Browser. Just log in at the normal WhatsApp Web, then open the development console. Now try entering something like the following:

```
var Wap = window.Store.Wap;
Wap .profilePicFind("91829xxxxxxx@c.us").then(res => console.log(res));
Wap .lastseenFind("91829xxxxxxx@c.us").then(res => console.log(res));
Wap .statusFind("91829xxxxxxx@c.us").then(res => console.log(res));
```

Using the Chrome developer console, you can see that Wap contains a lot of other very interesting functions. Many of them also return JavaScript promises. When you click on the *Network* tab and then on WS (websocket), you can look at all the communication between WhatsApp Web and its servers.

3.2 Chat identification / JID

The WhatsApp Web API uses these formats to identify chats with individual users and groups.

i. Chats: [phone number in Internation Format without +@c.us, e.g. 91xxxxxxxxxx@c.us when you are from India.
ii. Groups: [phone number of group creator]-[timestamp of group creation]@g.us, e.g. 91xxxxxxxxxx@g.us for the group that 91xxxxxxxxxx@c.us created.

iii. Broadcast Channels [timestamp of broadcast channel creation]@broadcast, e.g. 150998765@broadcast for a broadcast channel.

3.3 WebSocket messages

There are two types of WebSocket messages that are exchanged between server and client. On the one hand, plain JSON that is unambiguous and clear (especially for the API calls above), on the other hand, hard encrypted binary messages.

Unfortunately, these binary ones cannot be intercepted by using the Chrome developer tools.

3.4 Login and Encryption

WhatsApp Web encrypts the web socket data using several different algorithms. These include Curve25519 as Diffie-Hellman key agreement scheme, HKDF for generating the extended shared secret, AES 256 CBC and HMAC with SHA256.

Starting the WhatsApp Web session starts by just connecting to its web socket server at wss:// web.whatsapp.com/ws (wss:// means that the web socket connection is secure). Also, that connection can only be made by sending HTTP header Origin: https://web.whatsapp.com is set, otherwise the connection will be rejected or refused.

3.5 Messages

When messages are sent to a WhatsApp Web web socket, they need to be in a specific format. It is quite simple and looks like TAG,JSON, e.g. 1515598888,["dataToSend",123]. Note that message tag can be anything. This application mostly uses the current timestamp as a tag, just to be a bit unique. WhatsApp Web itself often uses message tags like s1, 1234.-

-0 or something like that. Obviously, the message tag should not contain a comma. Additionally, JSON response is possible as well as payload.

3.6 Logging in

For logging in, the following steps are followed:

i. Generate a new clientId, it needs to be 16 base64-encoded bytes (i.e. 25 characters). WhatsApp Web Application just uses 16 random bytes, i.e. base64.b64encode(os.urandom(16)) in Python.

ii. Decide for a TAG or messageTag for your message, which is more or less arbitrary. WhatsApp Application uses the current timestamp (in seconds) for that.

iii. The message you send to the websocket looks like this: TAG,,["admin","init",[0,3,417],["Chromium","Chromium Short"],"clientId",true].
   I. messageTag and clientId needs to be replaced.
   II. The [0,3,417] part specifies the current WhatsApp Web version. The last value changes frequently. It should be quite backward-compatible though.
   III. In your mobile device, Chromium will be shown in active WhatsApp clients

iv. Right after scanning, the web socket will receive a message in the specified format with the TAG or message tag. The JSON response of this message has the following attributes:
   I. status: should be 200 else something went wrong
   II. ref: in the application, this is treated as the server ID; important for the QR Code Generation.
   III. TTL: is 20000. It's the time after the QR code becomes invalid
   IV. update: a boolean flag indicating is an update required or already on the latest version
   V. curr: indicates the current WhatsApp Web version, e.g. 0.2.7314
   VI. time: indicates timestamp the server responded at, as floating-point milliseconds, e.g. 1515592038888.0

3.7 Generating QR Code

i. A private key is needed to be generated using Curve25519, e.g. curve25519.Private().
ii. Then Get the Public Key from that private Key in step 1, e.g. privateKey.get_public().
iii. Get the QR code string by concatenating these values with a comma:
    I. the server ID, i.e. the ref value
    II. PublicKey base64 encoded, i.e. base64.b64encode(publicKey.serialize())
    III. And your client ID
iv. Turn this string into an image using any library and scan it using the WhatsApp app.

3.8 Requesting New Ref for QR Code Generation

i. Up to 5 new server refs can be requested when previous one expires (TTL). Can be done by sending TAG,["admin","Conn","reref"].
ii. The server responds with JSON object with the following attributes:
    I. status: should be 200 (other ones: 304 - reuse previous ref, 429 - too many refs requested)
    II. ref: new value of ref
    III. TTL: expiration time
iii. Update your QR code with the new ref value.

3.9 What Happens after scanning QR Code

Immediately after the user scans the QR code, the web socket receives several important JSON payload messages that build up the encryption details. These use the specified message format and have a JSON response as payload. Their TAG has no special meaning. The first entry of the JSON payload has one of the following values:

i. Conn: array contains JSON object as the second element with connection information containing the following attributes and many more:
ii. battery: the current battery percentage of your phone
iii. browserToken: used to log out without active WebSocket connection
iv. clientToken: used for restoring/resuming closed sessions.
v. Phone: JSON payload detailed information about your phone, e.g. device_manufacturer, device_model, os_build_number, os_version
vi. platform: your phone OS, e.g. iPhone, Android
vii. pushname: Name user has set in their WhatsApp Profile
viii. secret
ix. serverToken: used for restoring/resuming closed sessions.
x. wid: User phone number with chat identification jid
xi. Stream: payload like ["Stream","update",false,"0.2.7321"]
xii. Props: array contains JSON object as the second element with several properties like imageMaxKBytes, maxParticipants, videoMaxEdge, and others.

3.10 Key generation for Encrypting and Decrypting Websocket Messages

i. It's time for generating the final encryption keys. First, the secret from Conn payload should be base64 decoded and stored as secret. This decoded secret will be 144 bytes long.
ii. Create Public key from the first 32 bytes of secret generated in step i. Together with the private key, generate a shared key out of it and call it sS. The application does it using privateKey.get_shared_key(curve25519.Public(secret[:32]), lambda a:a).
iii. Extend sS to 80 bytes using HKDF. Call this value sSE.
iv. HMAC validation is done, just to validate data received on WebSocket. The method is called HMAC validation.
v. Do it by first calculating HmacSha256(sSE[32:64], secret[:32] + secret[64:]).
vi. Compare this value to secret[32:64].
vii. If they are not equal, abort the login.
viii. You now have the encrypted keys: store sSE[64:] + secret[64:] as encryptedKeys.

ix. The encrypted keys now need to be decrypted using AES with sSE[:32] as key, i.e. store AESDecrypt(sSE[:32], keysEncrypted) as decryptedKeys.
x. The decryptedKeys variable is 64 bytes long and contains two keys, each 32 bytes long.
xi. Binary Messages sent to you by WhatsApp Web Server are decrypted using encKey and also used to encrypt binary messages that will be sent to the server.
xii. macKey is used to validate the messages, send by the user.
    I. encKey: decryptedKeys[:32]
    II. macKey: decryptedKeys[32:64]

3.11 Restoring Closed Sessions

i. Send the Init Command.
ii. Send TAG,["admin","login","clientToken","serverToken","clientId","takeover"].
iii. Replace serverToken and clientToken, with one already saved from the last session
iv. If server respond with {"status": 200} then its Ok, else:
    I. 401: Unpaired from the phone
    II. 403: Access denied, check to field in the JSON: if it equals or greater than 2, you have violated TOS
    III. 405: Already logged in
    IV. 409: Logged in from another location

3.12 Resolving Challenge

i. When one is using expired or maybe old serverToken and clientToken, a challenge needs to be solved in order to confirm that user still has access to valid encryption keys.
ii. Challenge payload looks like: TAG,["Cmd",{"type":"challenge","challenge":"bas64encodedString=="}]

iii. Decode the base64 encoded string then, sign it with your macKey, encode it back with Base64 and send:
TAG,["admin","challenge","bas64encodedString==","serverToken","clientId"]
iv. Server will respond with {"status": 200}, but it means nothing.
v. Once the challenge is solved, the connection will be restored.

3.13 Logging out

i. With an active WhatsApp Web socket connection, just send goodbye,,["admin","Conn","disconnect"].
ii. Send a POST request to:
https://dyn.web.whatsapp.com/logout?t=browserToken&m=logoutToken
iii. It's recommended to always clear sessions after use.

3.14 Validating and decrypting messages

There'll be two keys for validating and decrypting messages the server sent to user is quite easy. Note these keys are only needed for binary messages, all JSON payload you receive stays plain. First 32 bytes of the binary message is HMAC checksum, used for data validation. Both JSON payload and binary messages have a TAG at their very start that can be discarded, i.e. only the payload/data after the first comma character is significant.

i. Validate the binary message by hashing the actual message content with the macKey: HmacSha256(macKey, messageCt[32:]). If this value is not equal to messageCt[:32], the message sent to you by the server is invalid and should be discarded.
ii. Decrypt the binary message content using AES and the encKey: AESDecrypt(encKey, messageCt[32:]).

After decrypting user will get the final data in the readable binary payload.

3.15 Dealing with End to End Encrypted Media

i. Encryption:
   I. A 32-byte media key needs to be generated., which is then expanded to 112 bytes using HKDF. This is done with type-specific application info. This value will be denoted as mediaKeyExpanded.
   II. mediaKeyExpanded is split into IV[:16], cipherKey[16:48], macKey[48:80] and refKey[80:] (not used).
   III. The file is encrypted with AES-CBC with the help of cipherKey and IV. It is then padded. This will now be called enc.
   IV. IV and enc are signed with macKey using HMAC SHA-256. The first 10 bytes of the hash is stored as mac.
   V. Hash the file with SHA-256 and store it as fileSha256, hash the enc + mac with SHA-256 and store it as fileEncSha256.
   VI. The fileEncSha256 is encoded with base64 and stored as fileEncSha256B64.
   VII. For streamable media such as audio and video: Sign every [n*64K, (n+1)*64K+16] chunk with macKey and then truncate the result to the first 10 bytes. After that everything is combined.

ii. Uploading:
   I. The upload-URL is retrieved by sending messageTag["act", "enc_up", typeOfFile, fileEncSha256B64].
   II. The type of file can audio, image, video or document.
   III. A multipart-form is created containing the following fields:
   IV. hash: fileEncSha256B64
   V. file, filename: blob: enc+mac
   VI. A POST request is made to the URL with the query string ?f=j along with the content-type and the multipart-form. In response to the request, the URL for downloading the file is sent by WhatsApp.

Now, all the required information for sending the file has been generated. After this, build the proto and then send.

iii. Decryption:

I. The mediaKey is obtained. If required, it is also decoded from Base64.

II. It is then expanded into 112 bytes using HKDF along with type-specific application information. This is known as the mediaKeyExpanded.

III. mediaKeyExpanded is split into IV[:16], cipherKey[16:48], macKey[48:80] and refKey[80:] (not used).

IV. Media data is downloaded from the URL and then split into:

V. file: mediaData[:-10]

VI. mac: mediaData[-10:]

VII. Media data is validated with HMAC by signing Initialization Vector and file with macKey using SHA-256. One should keep it in mind that mac is truncated to 10 bytes. Therefore, the comparison should be made only with the first 10 bytes.

VIII. The file is decrypted with AES-CBC using cipherKey and Initialization Vector. After that, unpad it. This means that the session keys are not necessary to encrypt the data.

Chapter – 4

ALGORITHMS / WORKFLOWS

In this chapter, the algorithm/workflows of basic functionalities will be discussed.

4.1 Login

A WhatsApp Web like Login GUI has been created in Golang. User just needs to open the URL and it will load the QR code by following the implementation that was discussed earlier. Once the User scans the code, it will pop up an API Key / Username, that can be further used to access the API Functionalities.

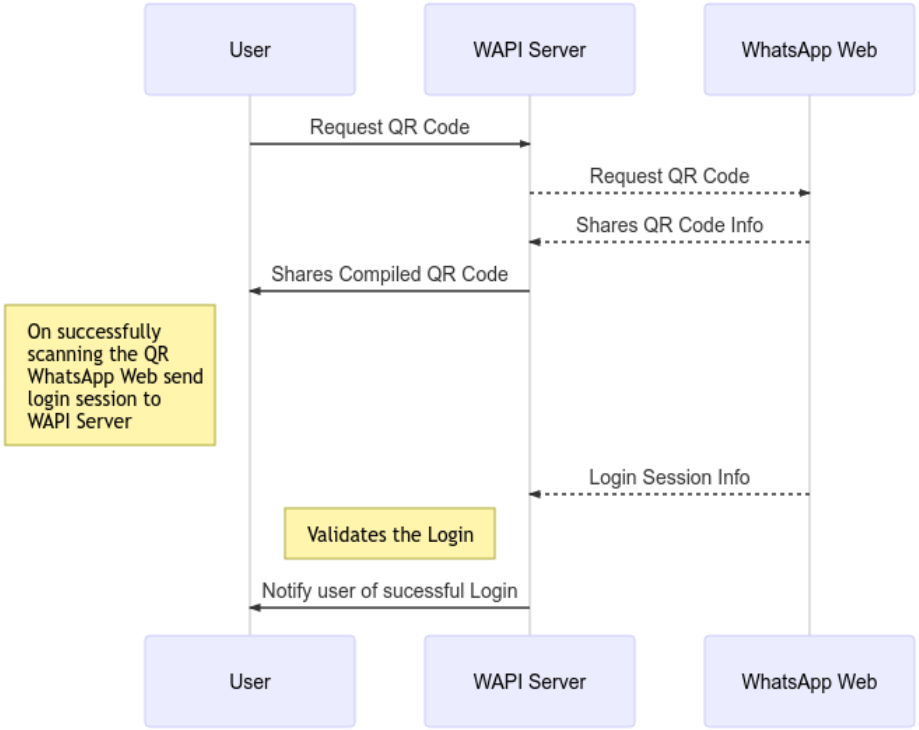

Fig. 4.1 WAPI Login Workflow

The steps involved are as follows:

i. User opens the application.
ii. Application requests the QR code from wapi rest api backend.
iii. The backend web socket generates the QR code.
iv. The QR code is then sent to the main screen.
v. The code is scanned by the user.
vi. Backend processes the further message and completes the login part till the creation of encryption and decryption keys.
vii. The keys are then stored in the session file.
viii. User is notified that login is successful and the username can be used to access all other api functionalities.

4.2 REST API

Further REST API service enables the user to access WhatsApp Functionalities like sending/receiving (text/image/audio/video/links/location/vCard/voice message), group creation and other manipulation.

Although it is not possible to compete with the Official Business API, but the API built, offers a lot of functionalities to get started with automating things and it also helps in creating chat bots.

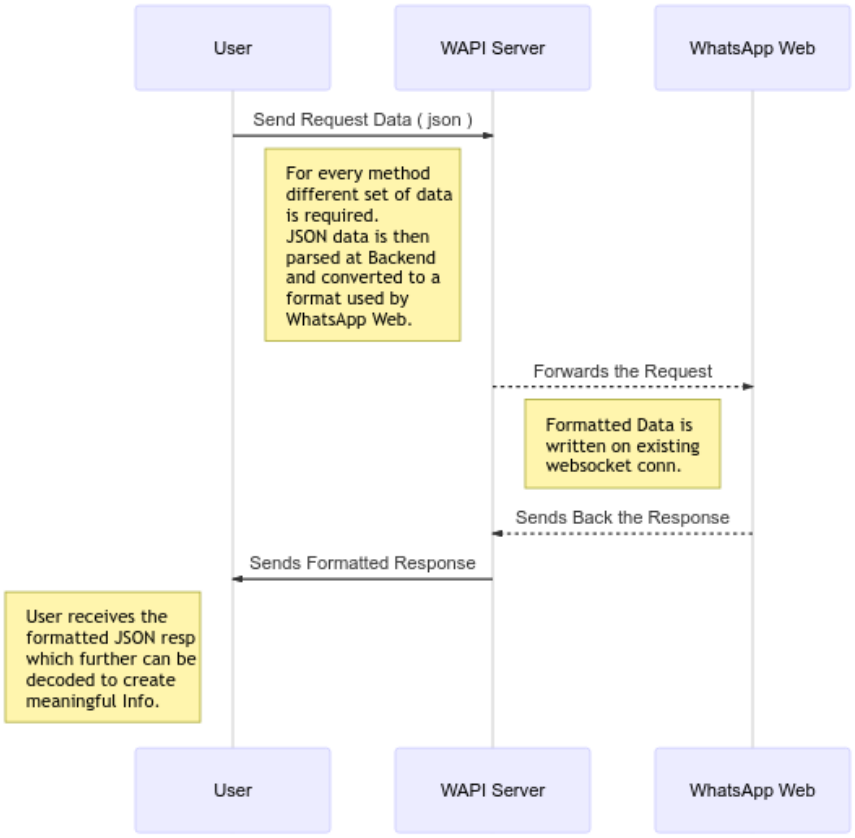

Fig. 4.2 RESTAPI Workflow

The steps involved are as follows:

i. Send request to API.
ii. API will load session details from cache and restores the session, if not already logged in.
iii. Formats the request in a way that WhatsApp Web understands/use.
iv. Wait for the response from WhatsApp Web.
v. Send the final formatted response to user.

4.3 WEBHOOK FUNCTIONALITY

REST API also includes some methods which lets you set your webhook URL, Enable the type of data you want to receive on that particular URL and webhook's can be customised to call another function against a particular set of received data. For Ex, you want to send a message with predefined text or maybe some algorithmic reply, whenever you receive a message on WhatsApp, you can do that with Webhook Functionality.

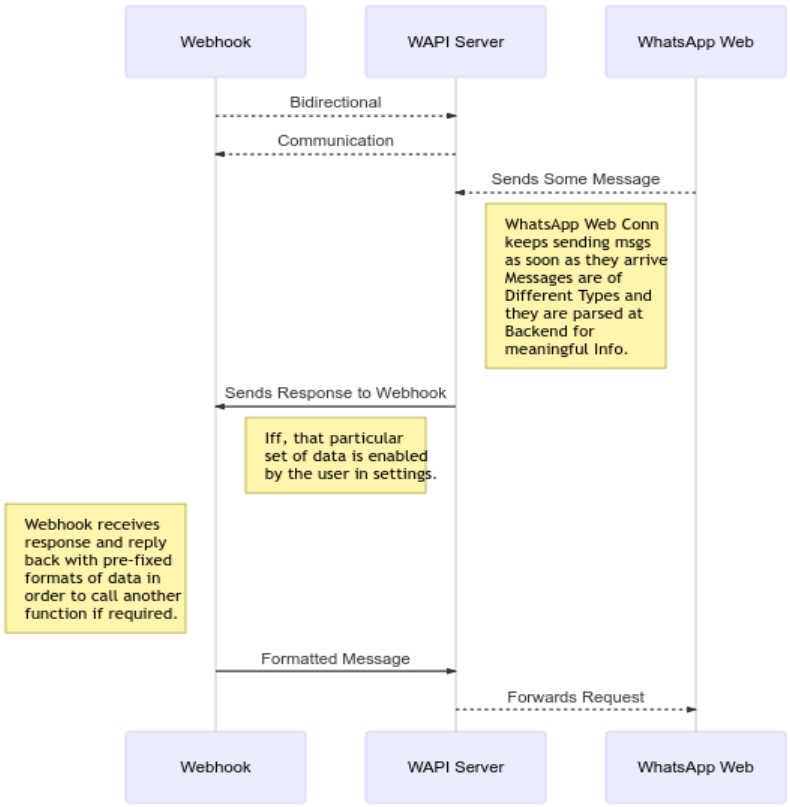

Fig. 4.3 Webhook Feature Workflow

The steps involved are as follows:

i. WAPI Server receives message from WhatsApp Web.
ii. If user has enabled settings for that type of data, it will be sent to webhook.
iii. Then webhook can process the data, and also reply with data with defined formats in order to send a reply (text/image/audio/video/links/voice messages etc.)
iv. If webhook replies with some data, data will be processed and respective functionality will be called with required data from webhook response.

With webhook feature, any kind of bots can be build, from simple Echo Bot to a Multipurpose WhatsApp Bot ☺.

Chapter – 5
CONCLUSIONS

5.1 Conclusion

WhatsApp Web Reverse Engineered API is a perfect alternate to Official WhatsApp Business API. Though it has some limitations, but it still fulfils the need for the WhatsApp API for Everyone.

Fig. 5.1 WhatsApp API for Everyone

This API took almost same time to send and receive messages as of Official API at an exceptionally lower cost. Official API costs some fixed amount for every message sent, and it also charges for receiving or pulling messages. Where on the other hand, this API costs nothing other than the infra cost we are deploying it too. (as per my tests and experiments, this API doesn't have any heavy resource requirements).

5.2 Future Scope

i. **WhatsApp based Mobile Number Verification:**

Traditional one-time password via sms can be replaced. Even it can be used for 2 factor authentication.

ii. **Courier Updates on WhatsApp:**

Better and more convenient way for customers to track on their packages

iii. **Invoices on WhatsApp:**

Paper Bills, Payment Slips all can be avoided, if they can be delivered on WhatsApp.

And it will be more convenient to keep a log of those receipts and slips, if they are available digitally.

iv. **RSS Feeds on WhatsApp:**

What's better than receiving your daily news crunch or favourite blog rss feeds on WhatsApp.

v. **Customer Care Service on WhatsApp:**

Will be a little complicated to implement, but at least customers will not be listening to those boring tunes while waiting for customer executive to connect and also tracking quality of chat is way easier than tracking quality of voice call.

vi. **Shopping Assistant:**

vii. **Personal Bots:**

N number of possibilities are there, Bot for Checking Server Health / Resource Usage, Bot for Managing / Tracking Expenses, To-Do / Reminder Bot, Easy Event Check-in Manager, Wikipedia / Dictionaries / News on WhatsApp etc.

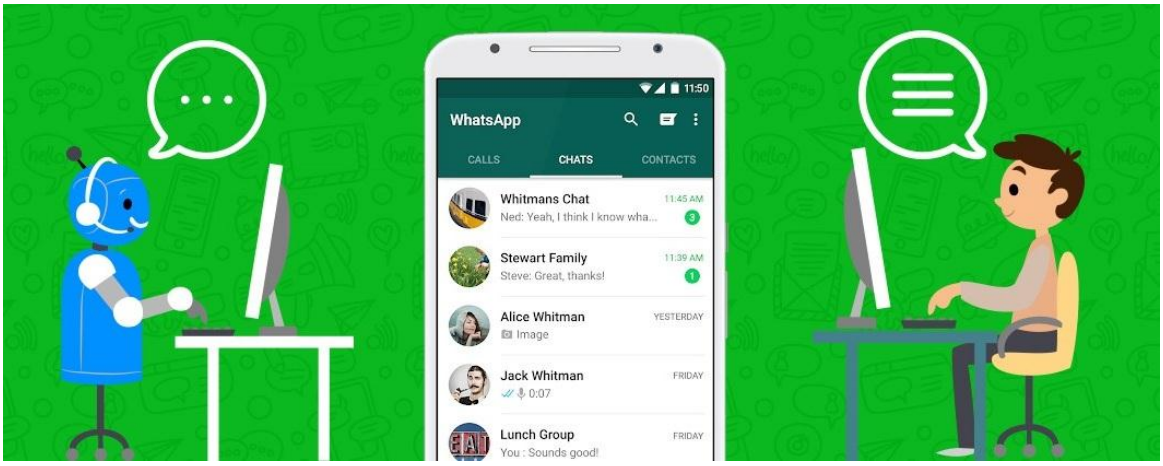

Fig. 5.2 WhatsApp Chatbot Future

We know, Chatbots are the future, they have already gained a good market share and with a good consistently growing user base, WhatsApp is here to stay.

So Integration of Both things, opens up thousands of new possibilities.

5.3 Applications

I have made the API service public at https://wapi.xyz

Following are some of the applications developed by some companies, individuals and me using the same API.

i. WhatsApp for Tally:

WhatsApp for Tally allows you to send any piece of document like bills, invoices, outstanding statements, ledgers and more from Tally directly to the end-user with ease.

Features:

i. Recipients mobile numbers get picked up automatically from the ledger.
ii. Unlimited Messages can be sent and logs for all of them can be accessed later.
iii. WhatsApp message can be sent from any screen in Tally.
iv. Messages can be sent from your own WhatsApp number.

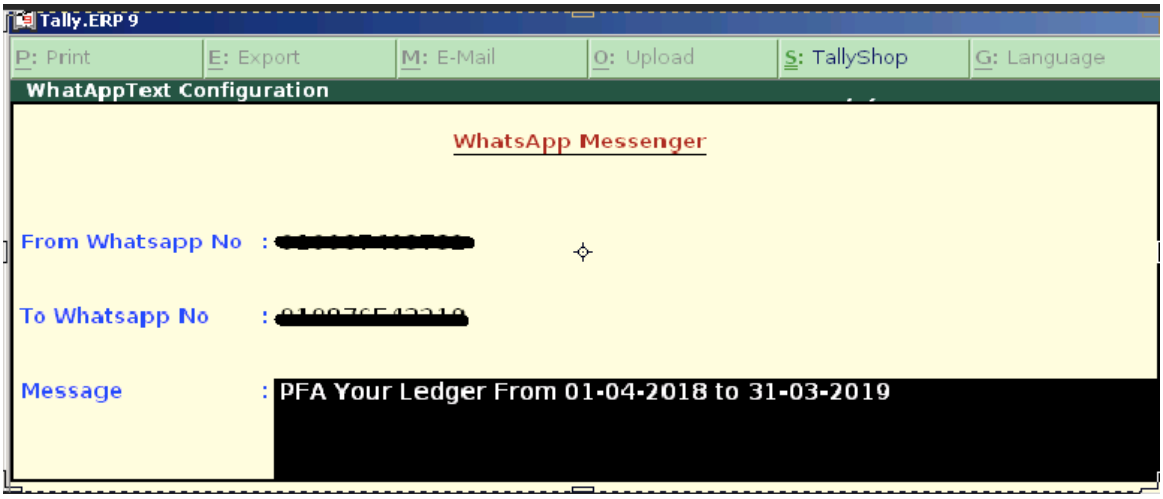

Fig. 5.3 WhatsApp Integration with Tally

ii. WhatsApp for Point of Sale Devices

WhatsApp for Point of Sales is similar in nature with the Tally Integration, but solve multiple purposes and also save paper. It allows Point of Sales owners to send Invoices / Payment Slips / Reminders / Offers on WhatsApp instead of sms or on paper slip. This particular model is running under opt-in behaviour, i.e. if a customer opts-in for WhatsApp messages instead of sms or hard paper slips, he/she will receive all of this on WhatsApp.

Features:

i. Allows customers to opt-in opt-out any time.
ii. Instant Delivery with more media types compared to traditional sms.
iii. Helps in saving paper, by not printing paper slips, but sending them on WhatsApp as a pdf.
iv. Makes tracking of payment slips / receipts easy at both customer and seller end.

iii. WhatsChannel - WhatsApp Based Broadcasting Service

WhatsChannel is a WhatsApp based Broadcasting Service (Unofficial) that can be used to send updates/newsletters/alerts to opt-in customers.

Though WhatsApp offers broadcasting service, but it has some limitations.
A broadcast can be send to 256 contacts only, and only contacts who has the sender number in their contact list will receive the message. Like if, selecting 256 contacts manually was not enough for problem sake.

What this service does is, allow anyone to create a Channel, once created creator will be added to a dedicated group with Channel provider, all the messages sent to that particular group, will be sent to all the subscribed users.
and for subscription, creator gets a short code and easy to share links, it can be shared with the directed audience or market. It simply sends a predefined subscription or subscription message to Channel Service provider.

Features:

i. Allows users to opt-in opt-out any time.
ii. Creator doesn't have to reveal its personal number; all the messaging will be taken care of by Channel Service provider.
iii. No more spamming in the name of Updates/Newsletters, user can opt-in and opt-out any time.
iv. No more 256 contacts limit, user base can go up to any limit.
v. No more time wasting on selecting contacts manually to broadcast message, Channel Service take cares of user management at backend
vi. Easy to Export User subscription list, if required in any case.

iv. WABot – Multipurpose WhatsApp Bot

WABot is a multipurpose WhatsApp Chat Bot, built for providing the most basic services on WhatsApp.

Features / Services:

i. **Conversation Chat Bot:**
   AI based conversational Chat Bot.

ii. **Wikipedia Search:**
   Search anything from Wikipedia, and get results in real-time

iii. **English Dictionary:**
   Just a dictionary but on WhatsApp.

iv. **Thought of the Day:**
   Get Latest Thought for the day, from some website.

v. **Horoscope:**
   Get your horoscope on WhatsApp.

vi. **Currency Rates:**
   Latest Currency Conversion Rates of USD, EUR, GBP, AUD and CAD to INR

vii. **Jokes:**
   Simple Jokes Service serviced by a Local Database.

viii. **Shayari:**
   Simple Shayari Service serviced by a Local Database.

ix. **Learn English:**
   Fetches New English Word every day, with meaning and pronunciation.

Note : All Services are linked with third party API or crawlers I made to fetch the related data.

Following are screenshots of WABot Services:-

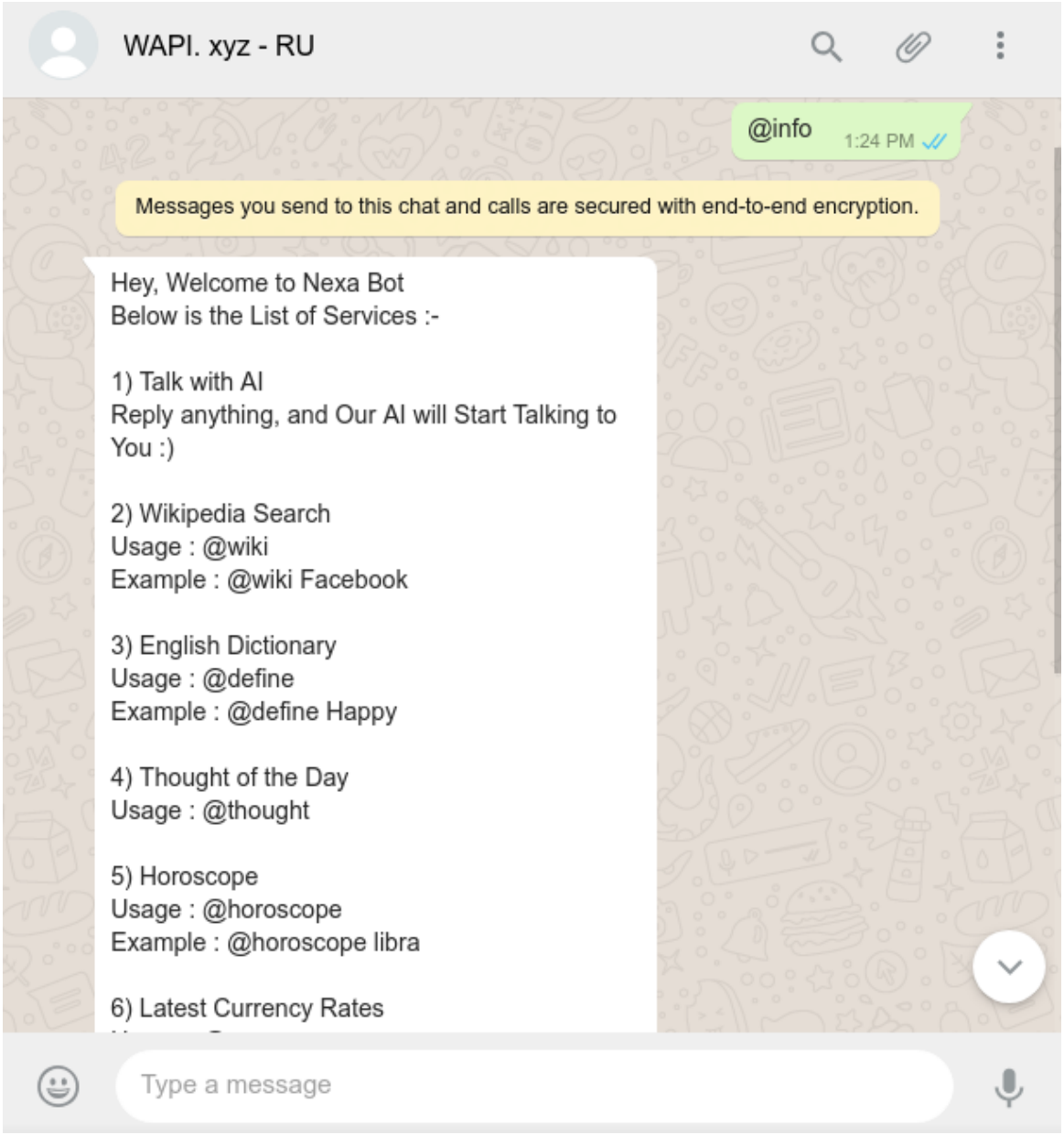

Fig. 5.4 WABot Info Message

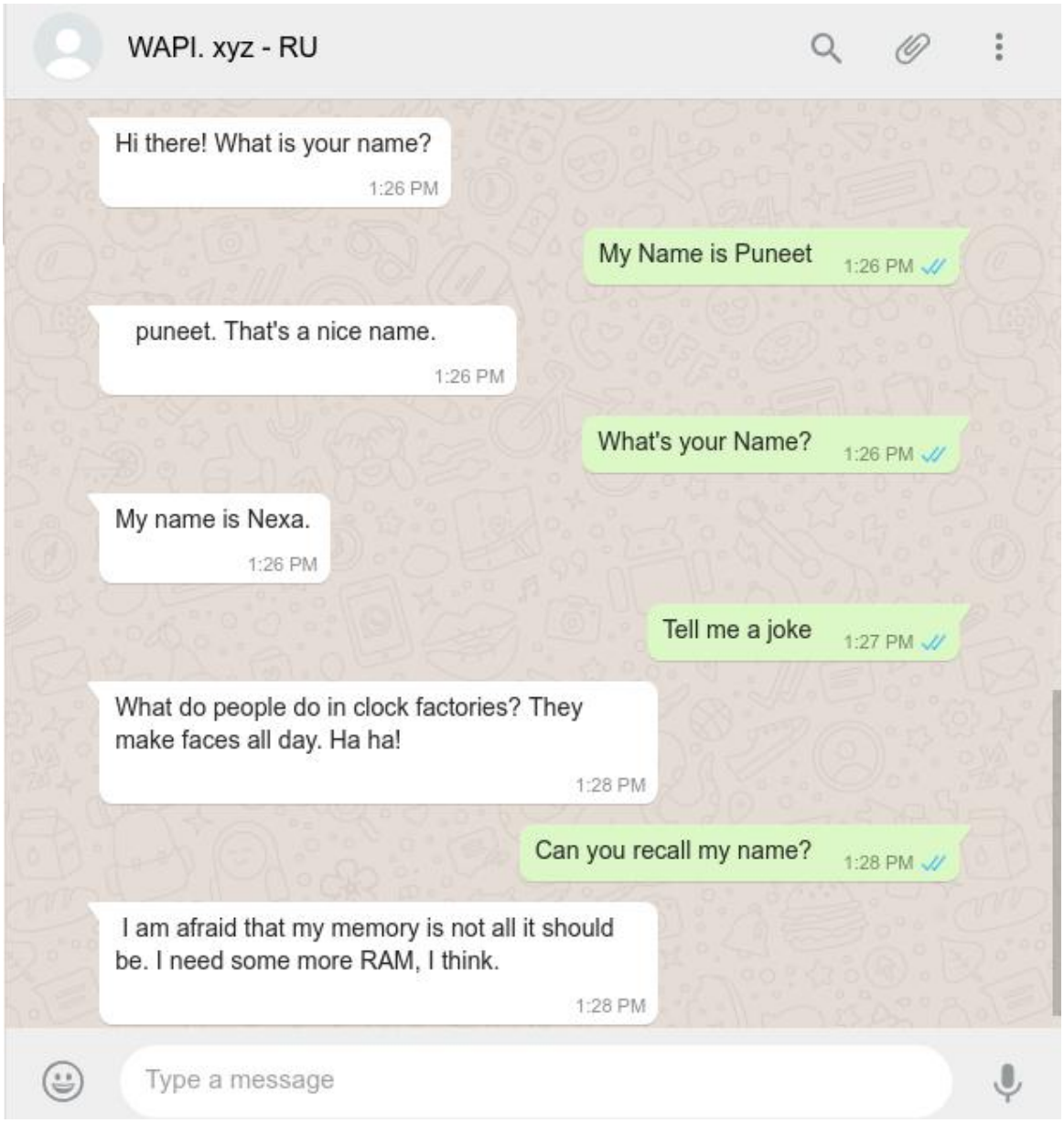

Fig. 5.5 WABot Conversational ChatBot

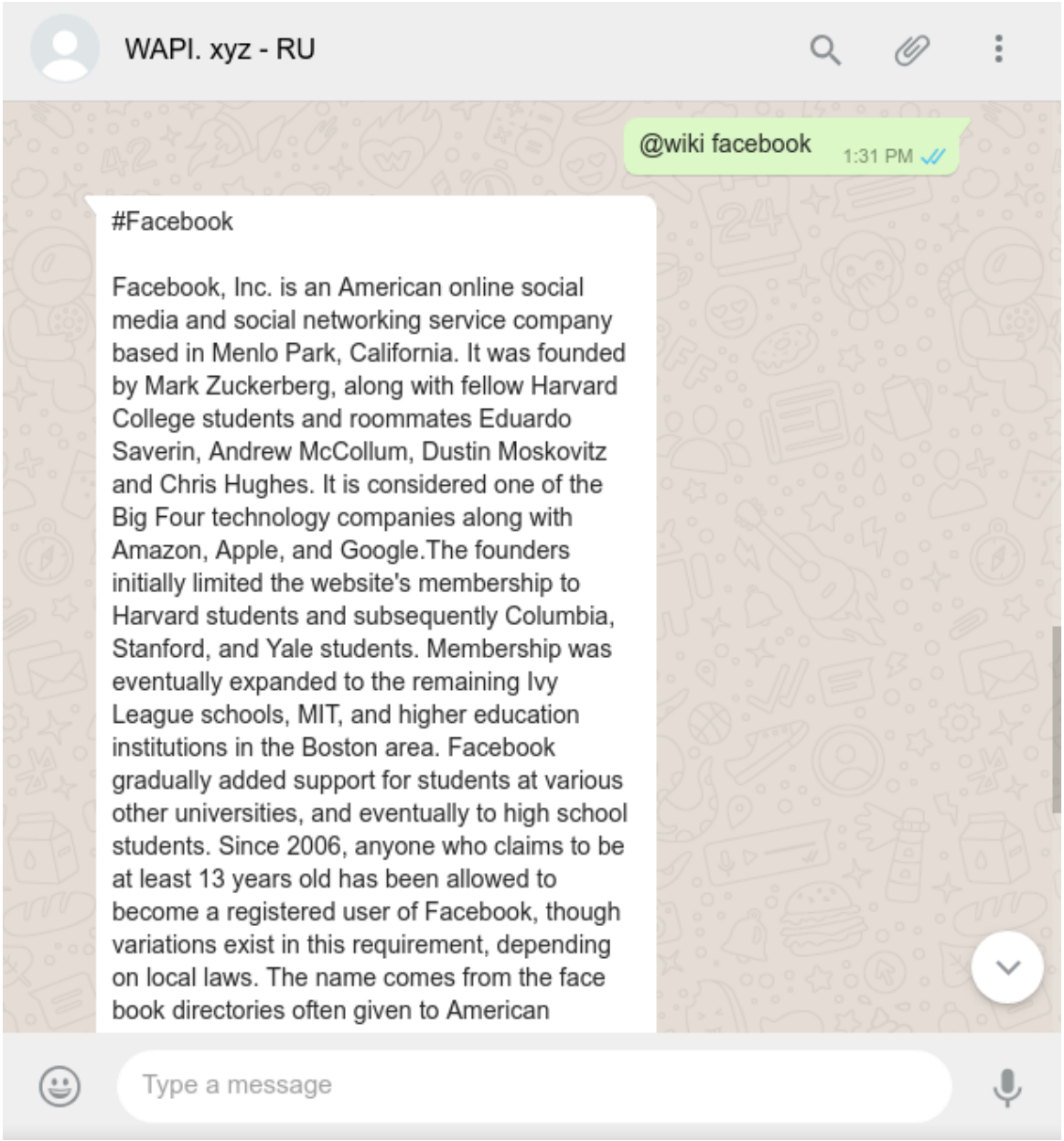

Fig. 5.6 WABot Wikipedia Search

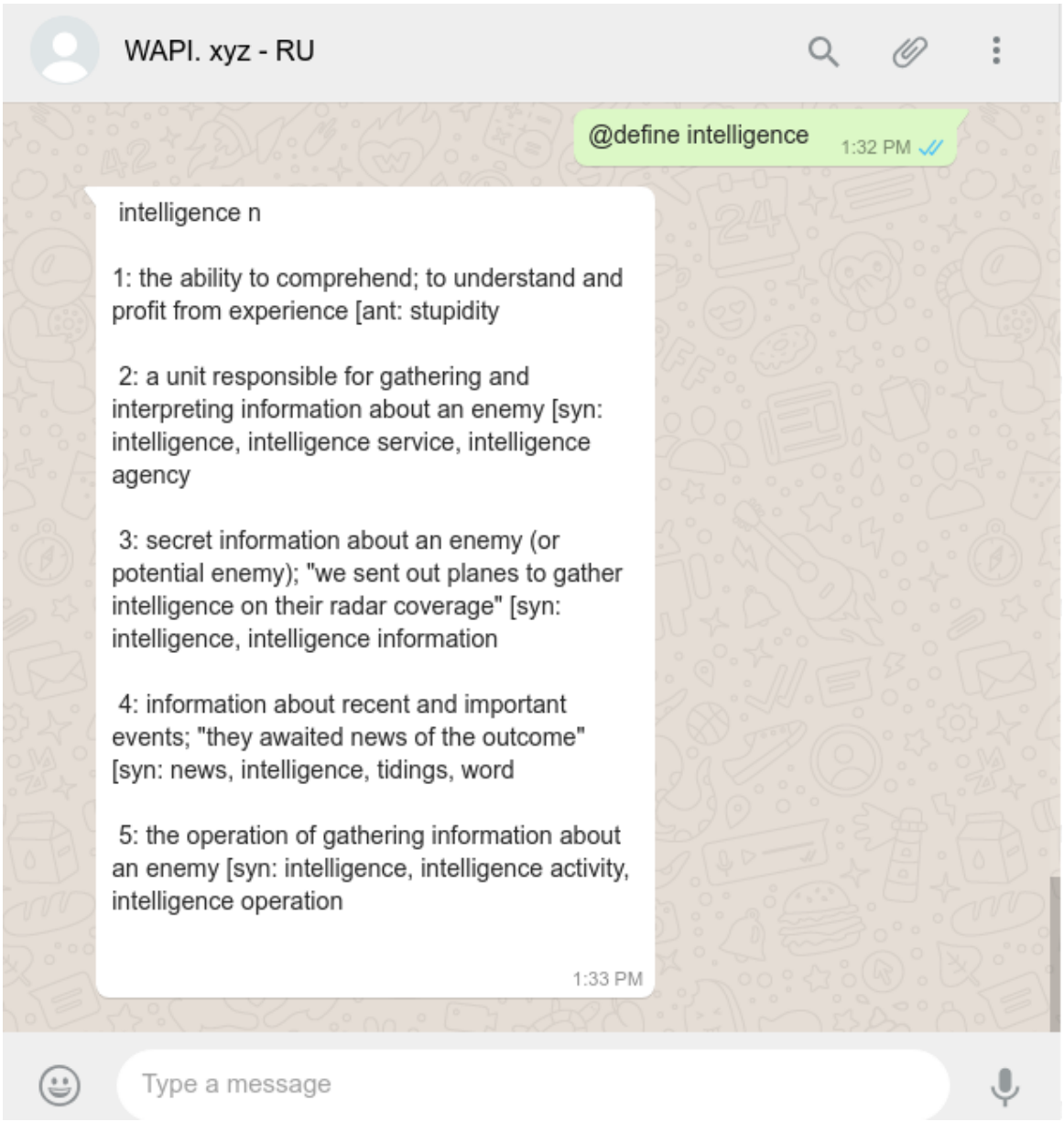

Fig. 5.7 WABot English Dictionary

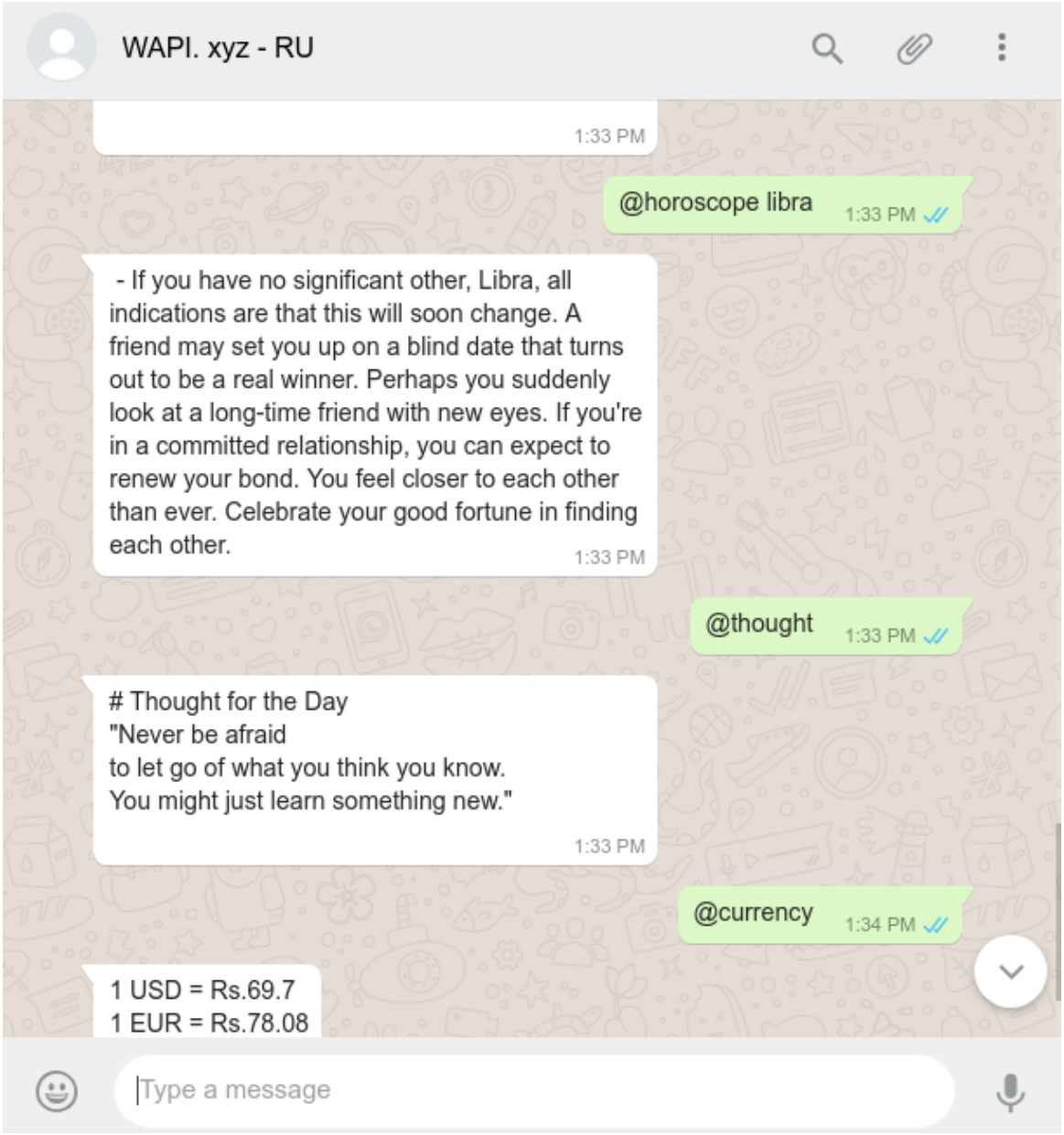

Fig. 5.8 WABot Horoscope, Thought of the Day and Currency Service

References:

[1] WhatsApp. “About WhatsApp” Accessed November 15, 2018. [Online]. Available: https://www.whatsapp.com/about/

[2] WhatsApp. “WhatsApp Features” Accessed November 10, 2018. [Online]. Available: https://www.whatsapp.com/features/

[3] WhatsApp Blog, “Connecting 1 Billion Users Every Day”, July 26, 2017. [Online]. Available: https://blog.whatsapp.com/10000631/Connecting-One-Billion-Users-Every-Day

[4] WhatsApp Blog, “WhatsApp Web”, January 21, 2015. [Online]. Available: https://blog.whatsapp.com/614/WhatsApp-Web

[5] Jan, Brian “end-to-end encryption”, WhatsApp Blog, April 5,2016. [Online]. Available: https://blog.whatsapp.com/10000618/end-to-end-encryption

[6] Bernstein, D. J. (2006, April). Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography* (pp. 207-228). Springer, Berlin, Heidelberg (pp. 5-6)

[7] WhatsApp, “WhatsApp Encryption Overview: Technical White Paper,” WhatsApp Inc., December 19, 2017. (Originally Published: April 5, 2016) [Online]. Available: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf

[8] WhatsApp Usage Statistics 2017 Available: http://fortune.com/2017/07/28/whatsapp-one-billion-daily-users/