

DMA Interoperability Developer Documentation: Overview of the Technical Framework

Important Information

This document provides a high level overview of WhatsApp's technical solution for facilitating interoperability pursuant to Article 7 of Regulation (EU) 2022/1925 (the "Digital Markets Act" or "DMA"). It constitutes part of WhatsApp's Reference Offer pursuant to Article 7(4) of the DMA.

Further detailed technical documentation, containing technical implementation guidance and a breakdown per technical requirement, will be shared with qualified partners.

Table of Contents

[Section 1 - Overview of Implementation and Milestones](#)

[Section 2 - Overview of WhatsApp's Technical Solution](#)

[Appendix: Required Features](#)

[Glossary](#)

Section 1 - Implementation and Milestones

WhatsApp has devised a sequenced, milestone-based approach to facilitate implementation of WhatsApp's technical framework for interoperability with eligible third party messaging services ("Partners"). The suggested sequencing below breaks down the implementation process into milestones that are accompanied by testable experiences. The testing enables the Partner to verify whether the implementation of each milestone component was successful.

Note: Partners can communicate with their WhatsApp point of contact for assistance while completing each milestone and when testing integration points.

Suggested Sequencing

There are **five main milestones** to implementing interoperability with WhatsApp. The breakdown below highlights the main activities to be completed in each milestone, as well as proposed testing to verify successful completion. Further specifics will be provided in the detailed technical documentation (which is organized by milestone).

Getting started

- Review the requirements and create a work plan

Milestone 1: Identities - Verification, User Enlistment and Authentication

- Build out the public key fetch APIs to allow the WhatsApp server to fetch public keys to verify user tokens
- Build and test user enlistment
 - Request and Responses

Test - enlist a user and verify successful response

Milestone 2: Chat Protocol and Noise Handshake

- Chat Channel / Noise Protocol
- Build out WhatsApp binary protocol -> XML translation to connect and see examples
- Build XML -> WhatsApp binary protocol translation and a ping; connect and send ping
- Build user key fetch XML construction & storage

Test - set up chat channel, complete noise handshake, verify successful responses received for ping

Milestone 3: Enabling Messaging

3.1 Encryption

- Build message encryption using the Signal Protocol (or compatible)

3.2 Enabling Outgoing Messages to WhatsApp

- Build message XML construction
- Build text message protobuf construction
- Send the phone numbers of WhatsApp test accounts you want to use for testing

Test - successfully send message from a Partner client to a WhatsApp client

3.3 Enabling Incoming Messages from WhatsApp

- Push notifications
- Build message XML processing
- Build message decryption
- Build text message protobuf parsing/validation

Test - successfully receive message from a WhatsApp client

Milestone 4: Enabling Integrity and Additional Features

4.1. Enabling Media

- Build and test subsequent message types, such as media

Test - successfully receive media from a WhatsApp client

4.2. Integrity Features

- Build blocking functionality

Milestone 5: User Deletion

- Partner enables and sends a deletion request through the chat channel to request removal of user account information from the WhatsApp server.

Test - send deletion request

Once all milestones have been reached, WhatsApp and Partner will jointly test the integration.

Conclusion: Enabling interoperability for Partner app

- WhatsApp and Partner to agree on rollout/GTM and ongoing monitoring

Section 2 - Overview of Technical Framework

This document presents a high level overview of the technical specifications for each component of WhatsApp's technical solution to enable interoperability, as follows:

- [Part 1: Identities - Verification, User Enlistment and Authentication](#)
- [Part 2: Chat Protocol and Noise Handshake](#)
- [Part 3: Enabling Messaging](#)
- [Part 4: Enabling Additional Message Features and Integrity](#)
- [Part 5: User Deletion](#)
- [Part 6: Updates and Ongoing Support](#)

Each part corresponds to a chapter of the detailed technical documentation, to be sent separately under NDA to qualified parties.

General Note on Architecture

Partner users' devices connect to the WhatsApp server using WhatsApp's proprietary XML protocol (based on XMPP), as described further below.

The WhatsApp server interfaces with Partner servers over HTTP in order to facilitate user authentication, push notification services, and management of Partner's users' media uploads. Partner's users' devices also send HTTP requests to the WhatsApp server in order to enlist.

Note: Supported HTTP Protocols - Partner servers must support both the HTTP/1.1 and HTTP/2 protocols. This allows WhatsApp's server to start with the simpler HTTP/1.1 protocol and switch to HTTP/2 later to improve efficiency if required.

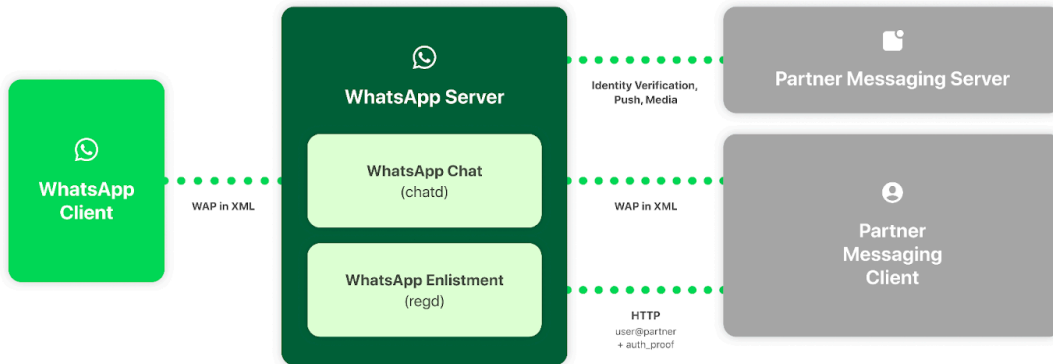


Figure 1.0: Solution Architecture Overview

Alternative Architecture (Proxy)

The Partner will have the option to add a proxy or an “intermediary” that sits between their client and WhatsApp’s server. The proxy server must maintain a **separate** client connection to the WhatsApp server for each individual Partner client on the Partner client’s behalf. The client/server protocol including the Noise Protocol can be implemented at the proxy level; however, the decryption & encryption of message content must *only* happen on the Partner client.

Requirements for Proxy Architecture

- The third party proxy server must maintain a single connection to WhatsApp Infrastructure per client
- The third party proxy server’s connection lifecycle must match that of the Partner client’s (i.e., the third party proxy server must not maintain a connection to the WhatsApp server while the device is offline)
- The third party proxy server must provide the client’s IP address to WhatsApp Infrastructure on connection
- The third party proxy server must not decrypt messages on the client’s behalf (i.e., breaking end-to-end encryption)

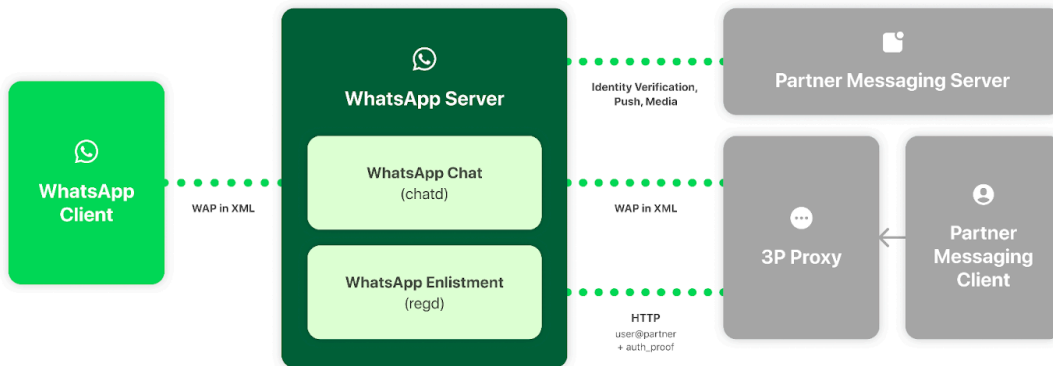


Figure 1.1: Solution Architecture Overview with Proxies

Part 1: Identities - Verification, User Enlistment and Authentication

WhatsApp's technical solution utilizes two main identifiers: a user-visible identifier and a uniquely generated identifier that is used at the infrastructure level (for protocols, data storage, etc.), referred to as an “internal identifier”.

Note: Requirements applicable to user-visible identifiers will be specified in the detailed technical documentation.

1.1 Verification of Identifiers

Each Partner user is represented by an internal identifier in WhatsApp’s infrastructure that is assigned to them upon enlistment.

When WhatsApp users message Partner users, the WhatsApp client refers to the internal identifier assigned to the Partner user at the protocol level. When Partner users

message WhatsApp users, the Partner client refers to the internal identifier assigned to the WhatsApp user.

WhatsApp requires Partner clients to provide “proof” of their ownership of the Partner user-visible identifier when connecting or enlisting. The proof is constructed by the Partner service, by providing a cryptographic signature over an authentication token, which proves that a device has access ownership to a given Partner user-visible identifier at a given point in time.

The WhatsApp server uses the *standard OpenID protocol* (with some minor modifications), alongside the JSON Web Token (JWT), to verify Partner user identities. When a Partner user attempts to enlist or connect to the chat channel, the WhatsApp server verifies their Partner user-visible identifier before performing the enlistment or opening the channels upon enlisting, as well as upon connection to the chat channel, as set forth in this section.

Partner servers are required to generate a signed token verifying the Partner user’s identity. The WhatsApp server will periodically fetch the public key from the Partner servers. The WhatsApp server uses this public key to verify the tokens generated by Partner, as follows:

- (1) The WhatsApp server periodically fetches the public key through an HTTP endpoint that Partner exposes. This public key is used for signature verification of the token.
- (2) The Partner server generates a token for the authenticated Partner client.
- (3) The Partner client provides that token to the WhatsApp server through either the enlistment or the chat channel connection points.
- (4) The WhatsApp server verifies the token by verifying the signature using the previously fetched public key.

1.2 Partner User Enlistment

The WhatsApp server exposes an Enlistment API that Partner clients must use when opting in to the WhatsApp network. The API consists of a standard HTTP request.

The Enlistment API has the following responsibilities:

- Verify the token, confirm that the provided Partner user-visible identifier is also mentioned in the token.
- Assign an internal identifier to the Partner client.
- Check the Partner client's WhatsApp interop protocol version number.

1.2.1 Enlistment Request Example

The Enlistment API Endpoint is: ***https://v.whatsapp.net/3p/interop_reg***

This API is executed by the Partner client to request to enlist the Partner user on WhatsApp's infrastructure.

After this request successfully completes, the Partner user can login to WhatsApp's chat service, known as chatd.

1.2.2 Arguments

Arguments should be passed into an HTTP POST request.

Arguments

Argument	Size	Description
authkey	32 bytes	Client static chat auth public key (same as for login). This key is saved on the server and used as part of the Noise handshake when connecting to the chat channel, a method of authenticating the client.
e_keytype	1 byte	e2e key type (0x05 == curve25519).
e_regid	4 bytes	The Registration ID RegistrationID is generated by the Signal Protocol
e_ident	32 bytes	e2e identity public key This is the long term key used to create Signal sessions
e_skey_id	3 bytes	e2e skey identifier

		Also referred to as the Signed PreKey ID, generated by the Signal Protocol
e_key_val	32 bytes	e2e skey bytes The public part of the Signed Prekey (a medium term key), generated by the Signal Protocol
e_key_sig	64 bytes	skey signature The signature over the Signed Prekey by the Identity Key
integrator_id		ID assigned to the Partner during onboarding e.g., "1" A base10 string representation (1-999)
user_identifier	1-4 bytes	The Partner user's user-visible identifier, e.g., "alice" or "16505551234". <ul style="list-style-type: none"> - Note: Each UTF-8 character is 1-4B - Restrictions: Only UTF-8 strings of length 1 to 40 are allowed
token	4 kilobytes	JWT token issued by Partner server

Enlistment Request Example

```
curl -X POST -k "https://v.whatsapp.net/3p/interop_reg" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  --data-urlencode "user_identifier=alice" \
  --data-urlencode
  "e_ident=x6g6tGpek00bx65-vVx9t3wNtjJUXcQY2iarR-Aush4" \
  --data-urlencode "e_keytype=BQ" \
  --data-urlencode "e_regid=x6g6tA" \
  --data-urlencode "e_skey_id=_tdJ" \
  --data-urlencode
  "e_skey_sig=BPEcejliI7RhacXe9MvBMUES_IHctlcDyza24vDrZca0N77YDGqCAXr2VWDALvOc
  TSziNWfVMJ4IDxIENwvyACA" \
  --data-urlencode
  "e_skey_val=_tdJvCOi0-F9h17iaHhrn42d1GnI3p6LwkFx3BS_jXw" \
  --data-urlencode
  "authkey=mLsPdd27k07BJa54dtt2e9Yz-vJfECLXRAPrFp7YeX4" \
  --data-urlencode "token=eyJraWQiOiIt..." \
  --data-urlencode "integrator_id=1"
```

Example Response (Successful)

Note: This is a JSON object.

```
< HTTP/1.1 200 OK
{"login":"1-10853634039893@interop", "status":"ok"}
```

Example Response (Failure)

Note: This is a JSON object.

```
< HTTP/1.1 200 OK
{"reason":"bad_integrator", "status":"fail"}
```

1.3 Fetch Public Key for Partner Identity Verification

The Partner client provides an auth token as the identity proof to WhatsApp server. WhatsApp server then cryptographically verifies if the identity does in fact belong to the Partner user in question, through the public keys fetched from the Partner server.

Fetch Public Key API Example

Sample Request

```
GET /.well-known/oauth/openid/jwks
```

Sample Response

```
{
  "keys": [
    {
      "kid": "370fa58a78d23043256a2cc580f1cf9ed3b9bbe7",
      "kty": "RSA",
      "alg": "RS256",
      "use": "sig",
      "n":
        "y1VSkbdJm-Q76nrPpv7RaGlQVjRfxw5z90wkM7qzFdUW15QJFrN8nozL21Vn0s-2rAfsoctoA
        8aQVMmtuWsyGIpdLrupMVskqC9bS5_7CQFXRGhQbxsZN1WDU9PcyzQamqFMfOE13Blj3gZr13T
        MHxa0B0IH60eMDlzUGWIsbrX4_QyYXr2Wj8xNNdZ7NwmRrik0dFTBg_-TjTvBzV10WkhbnHT7Lu
        kri1XSEKhjGqAeD0zfGaixhunx3PcFtuB5HNB9qdpkxMeDUG0oUZ3MX-WY5ppfYMBs8euR10PG2
```

```
6BAcBM5j40FuCy8zEv5ornodoa71Af-JbEdyr8smCyw",
  "e": "AQAB"
}
]
}
```

Part 2: Chat Protocol and Noise Handshake

WhatsApp utilizes a slightly modified version of the [Noise Protocol Framework](#). This protocol is used to encrypt all data traveling between the client and the WhatsApp server, similar to TLS.

As part of the Noise Protocol, the client must perform the Noise Handshake every time the client connects to WhatsApp's chat server. The Noise Handshake is used to establish a shared secret to encrypt the communication between the client and server. As the last part of the Noise Handshake, the client provides a payload to the server, which among many other data points also contains the JWT token.

2.1 Chat Protocol

Once the client has successfully connected to the WhatsApp server using the Noise Protocol, the client must use WhatsApp's chat protocol to communicate with the WhatsApp server.

WhatsApp's chat protocol uses XML to construct stanzas to communicate with the server. The data itself is transmitted in binary over the wire.

2.1.1 WhatsApp's Chat Protocol Stanzas

Some example stanzas include the following (*the detailed technical documentation includes more detailed stanzas*):

- <message> To send a message to a WhatsApp user.
- <iq> Similar to HTTP requests, used to query or set information with the WhatsApp server.
- <ib> For the server to communicate general information to the client, e.g., count of messages in the offline mailbox for the client.
- <receipt> Used by recipients of a message to indicate the status of the message to the sender (delivered, requires re-delivery).

2.1.2 WhatsApp's Binary Protocol Example

The WhatsApp binary protocol is used specifically in WhatsApp's chat protocol. The following is an example of how a stanza in WhatsApp's chat protocol is converted into the binary protocol.

XML:

```
<message to='14155551001@s.whatsapp.net' from='14155551000@s.whatsapp.net'
id='123456' type='text' notify='joe'>
  <body>Body text of message.</body>
</message>
```

List format:

```
['message', 'to', ('14155551001', 's.whatsapp.net'), 'from',
('14155551000', 's.whatsapp.net'), 'id', '123456', 'type', 'text', 'notify',
'joe', [['body', 'Body text of message.']]
```

Binary encoded with nibble & hex packing:

Note: The binary will be compressed and encrypted following this step.

```
[248, 12, 19, 17, 250, 255, 134, 20, 21, 85, 81, 0, 31, 3, 6, 250, 255,
134, 20, 21, 85, 81, 0, 15, 3, 8, 252, 6, 49, 50, 51, 52, 53, 54, 4, 56,
24, 252, 3, <<"joe">>, 248, 1, 248, 2, 237, 117, 252, 14, <<"Body text of
message.">>].
```

Part 3. Enabling Messaging

WhatsApp clients use the Signal Protocol to implement end-to-end encryption. Partner clients must also use this protocol (or one that Partner can demonstrate provides an equivalent level of encryption) in order to send and receive messages with WhatsApp clients.

As mentioned in the Enlistment API's encryption parameters, the Partner client will need to generate an identity key as well as some other keying material to be able to

send and receive messages to WhatsApp clients. This keying material is generated using the Signal Protocol.

Once the keying material is generated and the public keys are uploaded to the WhatsApp server, the Partner clients are ready to receive messages from WhatsApp clients. To send a message, they must use the chat channel and fetch the keys of the relevant WhatsApp user to encrypt a message to them.

Note: Only one encryption identity can be associated with a Partner user account at any given time. Therefore, WhatsApp clients only encrypt messages to a single device.

Review the *WhatsApp Security Whitepaper*, available publicly on the WhatsApp website, for more technical information about messaging security and encryption in WhatsApp.

3.1 Enabling Outgoing Messages to WhatsApp

To create outgoing messages, the Partner will need to build message XML stanzas and the content protobuf structures that are then later encrypted using the Signal Protocol.

More details are provided in the technical documentation.

3.2 Enabling Incoming Messages from WhatsApp

Partner clients must be connected to the WhatsApp server to receive messages from WhatsApp users. When the client is connected, messages are pushed by the server to the client through the chat channel, using the <message> stanza as defined in Part 2 above.

Once the message is received through the chat protocol, the client must decrypt the payload using the Signal Protocol or equivalent encryption protocol. Once the payload is decrypted, the content protobuf can be accessed and the message can be rendered.

3.2.1 Push Notifications

In the event that a Partner client is offline, the WhatsApp server will store the encrypted messages sent to the user for 30 days in their offline mailbox. In order to facilitate timely delivery of messages, the WhatsApp server will notify Partner servers over HTTP

that their user's client should reconnect to the WhatsApp server and fetch the latest messages waiting for them in their offline mailbox.

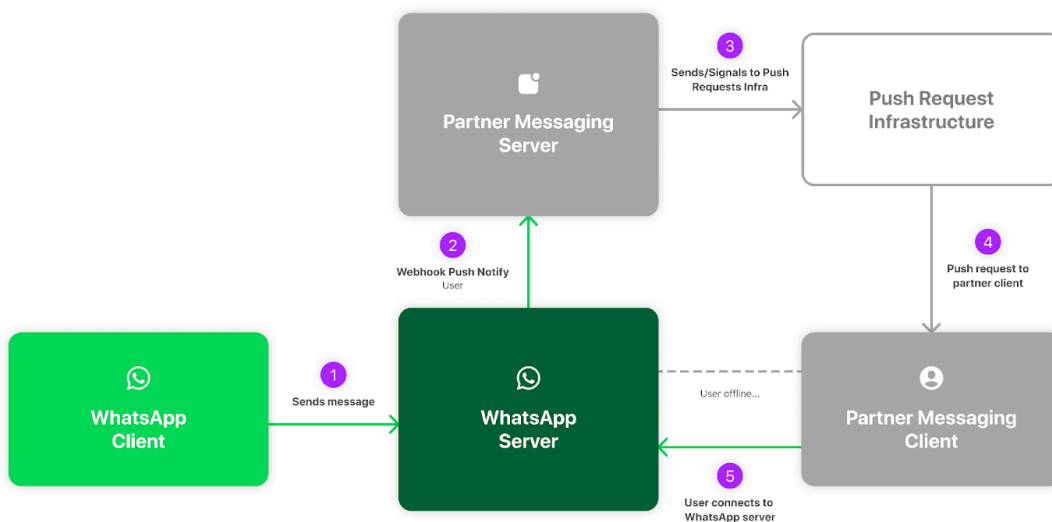


Figure 1.2: Push Notification Process Flow

3.2.2 Push API Example

Whenever there is new activity for the Partner user and the Partner client is not connected, the WhatsApp server makes a HTTPS POST request to the /push path (e.g., <https://foobar.com/whatsapp/push>) with the user-visible identifier as the payload.

Sample Request

```

curl --request POST
--url $BASE_URL/push \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer
eyJhbGciOiJIUzUzIiwiaXNjaW50IjoiImtpZCI6IjFhWGY4IiwidHlwIjoiSldUIiwiaWF0Ijoi.eyJpc3MiOiJXSEFUU0F
QUUCIsImF1ZCI6Imh0dHBzOi8vaW50ZWdyYXRvci5jb20iLCJpYXQiOiJlY20Dg2NjkyMzIsImV4cC
I6MTY4ODY3MjgzMn0.1JU_8k9SpbTo-MdGPdMqWNPdXR1NBwYU3HgkDJmHC9FDg6MQgwEByU7hy
-o2Wo_a09kf8p1JKYJQoxzbGsBfq3uAnN4ixKN2TsraV5VxkBTXjhDHRdifYQ1Zk0K1EQ501K5o
V0eU_WWjG7ykkb69AsIbw-phF__-t__zxD8t2fI6TLsq0j3i3zxIBLnR1kuvuHL2DR21JFV74Tm
AXPxeZQfxLrv542xV7C1J-ruBGS2hcb04M_OVvLbyXBPzP8M6n11hc460zSTmasTnzx1Etb7pGF
  
```

```
sn5eJwZVewWUQ3mbwDS7aTaC538PoA_EX8TpLMqyg05KkTGngKe8gygwcPFg'  
--tlsv1.3  
--http1.1  
--data '{"to": "alice421"}'
```

Sample Response

```
200 OK
```

Note: There is no payload expected in the response.

The authorization token is a JWT signed using WhatsApp server's private key. The Partner server should verify it using the public keys downloaded from <https://whatsapp.com/.well-known/jwks>.

Part 4. Enabling Integrity and Additional Features

To enable integrity features and additional features such as media management, the Partner will need to build and test media message types and blocking functionality.

4.1 Media Management

Partner messaging servers are responsible for hosting any media elements their client applications send to the WhatsApp client application (such as image files). The WhatsApp client downloads media from the Partner messaging servers using a WhatsApp proxy service.

Note: Partner client applications will download media from the WhatsApp server for media messages that are sent by the WhatsApp client application.

4.1.1 Sending Media to WhatsApp Client

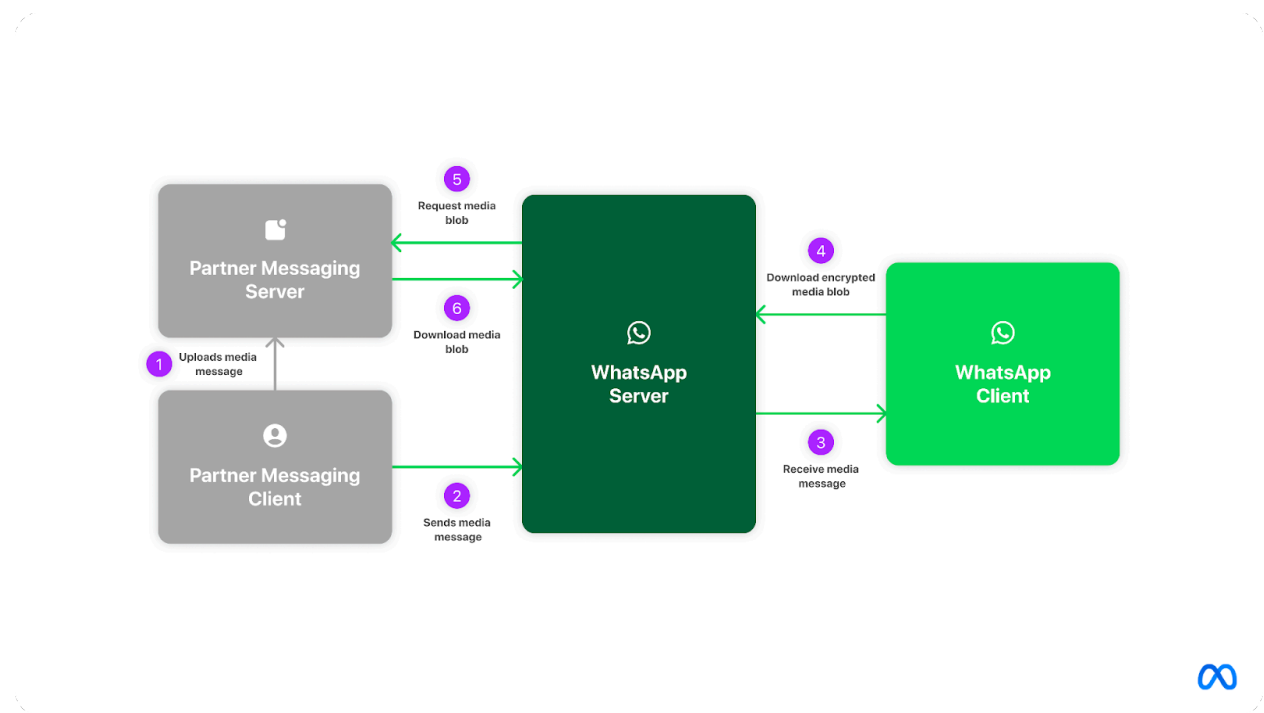


Figure 1.3: Partner Client to WhatsApp Client Media Flow

1. Partner clients should upload the encrypted media to the Partner server, obtaining a `storage_reference`.
2. Partner clients should send a media message with a download path as `/interop/whatsapp/download/{integrator_identifier}/{storage_reference}/`.
3. Upon receiving the media message, the WhatsApp client will construct a download URL by prepending the WhatsApp media server's URL to the download path (e.g., `http://mmg.whatsapp.net/{download_path}`) which is effectively:
`http://mmg.whatsapp.net/interop/whatsapp/download/{integrator_identifier}/{storage_reference}/`.
4. The WhatsApp client makes a download request at the above path to the WhatsApp server.
5. The WhatsApp server then downloads the encrypted media from the Partner server by making a GET request to `{3p_base_url}/{storage_reference}`, and returns a response to the client.

4.1.2 Receiving Media from WhatsApp client

The WhatsApp client will upload encrypted media to WhatsApp's media server to obtain a download path (i.e., `/v/t{file_identifier}`). The WhatsApp client will then pass the download path inside the media message.

The Partner client should construct the full download URL by prepending WhatsApp media base_url for download (e.g., `http://mmg.whatsapp.net/v/t{file_reference}`). The Partner client should then make a HTTP GET request to the endpoint (see example below).

Sample Request

```
GET http://mmg.whatsapp.net/v/t{file_reference}
```

Sample Response

```
200 OK
```

Note: The response header includes the content length (file size) and content type. The response body includes the file data binary.

4.2 Blocking

To block a WhatsApp user or to read the current list of users that the Partner client has blocked, getting and setting blocklist IQs are used over the chat channel.

Part 5. User Deletion

In the case of a Partner user deciding to opt out of the WhatsApp integration, the Partner client must send a deletion request through the chat channel to request removal of their information from the WhatsApp server.

If a Partner client does not connect to the WhatsApp server for a period of 30 days, the WhatsApp server will automatically un-enlist them. In that case, the Partner client will receive an error, and must re-enlist as described previously.

Part 6. Integration Completion, Ongoing Communication & Support

WhatsApp and Partner will jointly develop a rollout plan once implementation is successfully completed.

Partners will have access to a WhatsApp point of contact for ongoing communication and support post-rollout.

Versioning

The WhatsApp interop protocol supports versioning.

Partner clients are required to provide the protocol version they are using when connecting to the WhatsApp server.

Partners can refer to the version changelog that is published with the detailed technical documentation. The changelog will list any modifications that have been made to processes, endpoints, and other technical implementation details (such as protocol enhancements).

List of Required Features

The list below outlines the minimum set of features that Partners are required to support when requesting integration with WhatsApp for the Interoperable Messaging Services, as defined in the Reference Offer and described in this Developer Documentation.

Partner must support the following Required Features:

Feature Type	Feature List
User Features	<ol style="list-style-type: none"> 1. Receiving Messages <ul style="list-style-type: none"> - Text Messages - Media Messages 2. Sending Messages <ul style="list-style-type: none"> - Text Messages - (optional) Media Messages 3. Delivery Receipts <ul style="list-style-type: none"> - Must display to Partner User 4. Push Notifications
E2EE Features	<ol style="list-style-type: none"> 1. Security Notifications in the chat thread (i.e., Identity Change Notifications) 2. Forward Secrecy 3. Post Compromise Security 4. Deniability 5. Retry Messaging (i.e. receive and send retry receipts)
Processes and Functionality	<ol style="list-style-type: none"> 1. User Enlistment 2. Verification of Identifier for Partner Users 3. Deletion (user account deletion, e.g., upon opt-out)
Integrity and Safety Features	<ol style="list-style-type: none"> 1. Blocking

Glossary

Chat channel - The communications channel established and maintained between WhatsApp and Partner messaging services. The channel is between the client and the server (based on chatd).

Binary protocol - The transport protocol on the wire between the client and Chat server; the Chat protocol is encoded into this protocol.

Chat protocol - The client/server protocol used to communicate with WhatsApp's Chat server.

Devices - End user mobile devices (iOS and Android) that are used when sending and receiving messages between WhatsApp and Partner messaging services.

Enlistment (Registration) API - The set of Application Programming Interface calls that manage the messaging user enlistment process.

Infrastructure - The hardware and software resources comprising the Interoperable Messaging Services (servers, API endpoints, etc.).

Internal identifier - A unique identifier for users of the Partner messaging service created and used by WhatsApp. Used at the protocol layer.

JWT (JSON Web Token) - An open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. (See <https://auth0.com/learn/json-web-tokens>)

Noise Handshake - The exchange of public keys and the performing of a sequence of operations to compute shared secret keys independently by the WhatsApp's Chat server and Partner clients.

Noise Protocol - A framework for building crypto protocols. Noise protocols support mutual and optional authentication, identity hiding, forward secrecy, zero round-trip encryption, and other advanced features.

OpenID protocol - An open authentication protocol (OpenID Connect (OIDC)) that works on top of the OAuth 2.0 framework. OIDC allows individuals to use single sign-on to access relying party sites using OpenID Providers. (See <https://www.pingidentity.com/en/resources/identity-fundamentals/authentication-authorization-standards/openid-connect.html>)

Signal Protocol - A set of cryptographic specifications that provides end-to-end encryption for private communications.

User-visible Identifier - A unique identifier for a user of the Partner messaging service (phone number, email, etc.). Displayed to the end users.

User Enlistment - The process of setting up a new Partner user on the WhatsApp server.

WhatsApp Chat Server - The WhatsApp server that sends and receives messages between the WhatsApp client and Partner messaging applications.

WhatsApp Enlistment Server - The WhatsApp server that manages the client application and Partner user enlistment on the WhatsApp messaging platform.

WhatsApp Client - The WhatsApp client messaging application that runs on mobile devices (iOS and Android).

XMPP - The Extensible Messaging and Presence Protocol, a set of open technologies for instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalized routing of XML data. (See <https://xmpp.org/about/technology-overview/>)