# DMA Interoperability: Overview of the Technical Framework

**Important Information**

This document provides a high level overview of WhatsApp's technical solution for facilitating interoperability as required by Article 7 of Regulation (EU) 2022/1925 (the "Digital Markets Act" or "DMA").

As a general overview document, it is not the final authoritative source of Whatapp's final technical solution, which remains "work in progress" and subject to change.

Further detailed technical documentation, containing technical implementation guidance and a breakdown per technical requirement, is not included herein and will be shared at later stages.

Additionally, this document does not constitute WhatsApp's or Meta's Reference Offer required by Article 7(4) of the DMA.

***This document not to be shared externally.***

## Table of Contents

# Section 1 - Implementation and Milestones

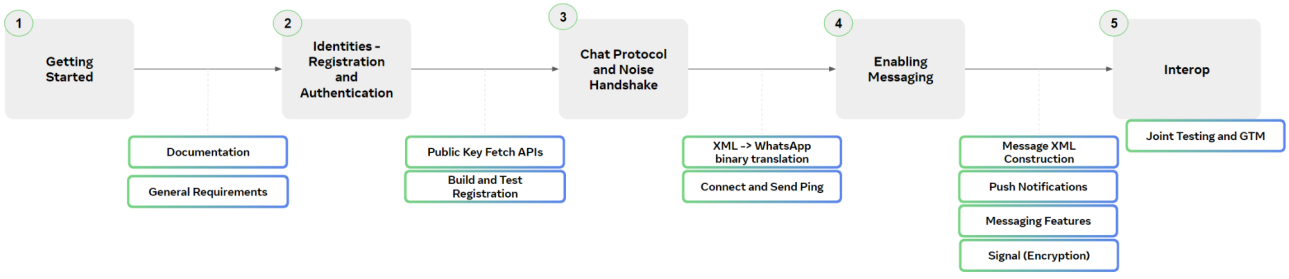
WhatsApp ("WA") has devised a sequenced, milestone-based approach to facilitate implementation of the proposed technical framework.

This sequenced approach to implementation is suggested only; the third party messaging service desiring to interoperate with WhatsApp ("third party") has flexibility to approach implementation as preferred.

By following the suggested sequencing below, a third party can break down the implementation process into milestones that are accompanied by testable experiences. This process improves eventual communication with WhatsApp for assistance and enables the third party to verify (through testing) whether the implementation was successful.

**Note:** The third party can communicate with their WhatsApp point of contact for assistance while completing each milestone and when testing integration points. Support program details for initial third party messaging partners is currently under review and subject to change.

## 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 - Registration and Authentication**

- Build out the public key fetch APIs to allow WA servers 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 -> XML translation to connect and see examples
- Build XML -> WhatsApp binary translation and a ping; connect and send ping
- Build user key fetch XML construction & storage

**Test** - setup chat channel, complete noise handshake, verify successful responses received for ping & key fetch requests

**Milestone 3: Enabling Messaging**

4.1. Encryption

- Build message encryption using the Signal Protocol.

### 4.2 Enabling Outgoing Messages to WhatsApp

- Build message XML construction
- Build text message protobuf construction
- Register test WhatsApp numbers in the portal or move integration to "onboarding"

**Test** - successfully send message from a third party client to a WhatsApp client

### 4.3 Enabling Incoming Messages from WhatsApp

- Push notifications *(optional)*
- Build message XML processing
- Build message decryption
- Build text message protobuf parsing/validation

**Test** - successfully receive message from a WhatsApp client

## Milestone 4: Enabling Additional Message Features + Integrity

### 5.1. Enabling Media

- Build and test subsequent message type, like Media

**Test** - successfully receive Media from a WhatsApp client

5.2. Integrity Features

- Build blocking functionality
- Build reporting functionality *(optional)*

**Milestone 5: User Deletion**

- Third party sends a deletion request through the chat channel to request removal of their information from WhatsApp servers.

**Test** - Send Deletion Request

**Conclusion: Enabling Interop for third party App**

- WhatsApp and third party to agree on rollout/GTM and ongoing monitoring
- Third party to request WhatsApp to enable interop for the app - **Production**

# 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: How Verification, User Enlistment and Authentication Work
- **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 at later stages *(Work In Progress)*.

## General Note on Architecture

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

WhatsApp servers interface with third party integrator servers over HTTP in order to facilitate user authentication, push notification services, and management of the third party's user's media uploads. Third party devices also send HTTP requests to WhatsApp enlistment service in order to enlist.

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

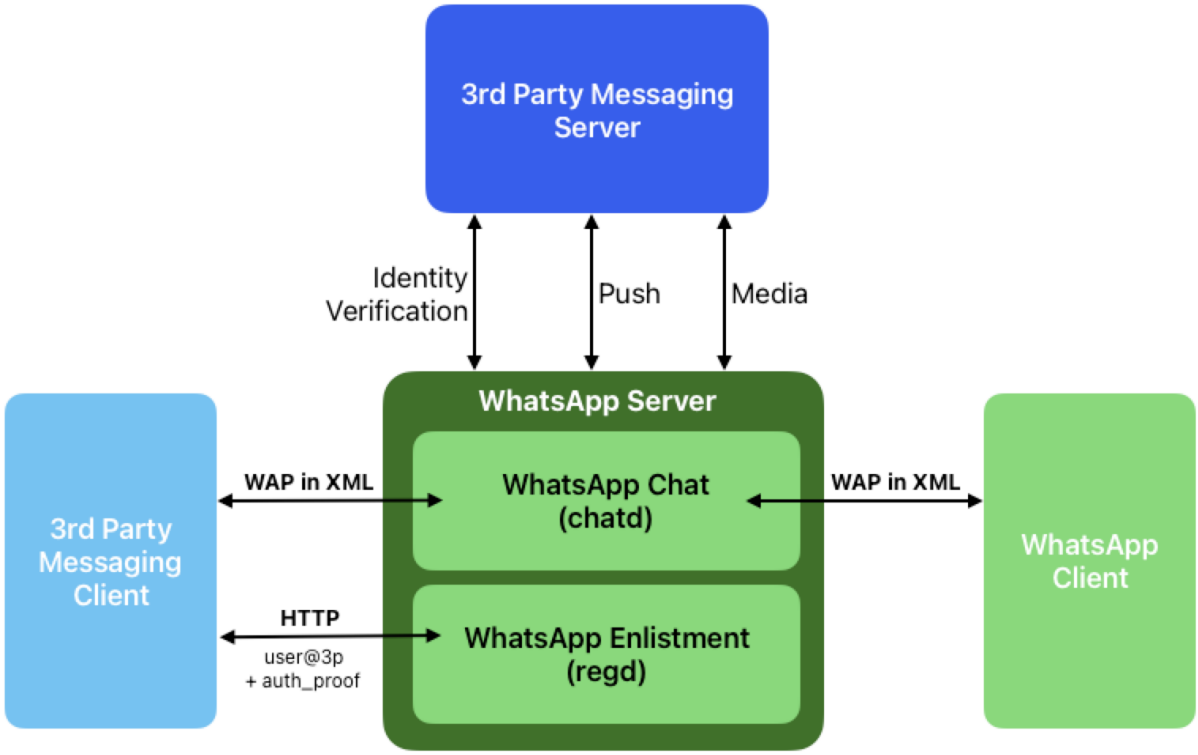


**Figure 1.0:** Solution Architecture Overview

# Part 1: Identities: How Verification, User Enlistment and Authentication Work

WhatsApp's infrastructure has two main identifiers: a user-visible identifier and another identifier that is used at the infrastructure level (for protocols, data storage, etc.), referenced as an “internal identifier”. When a third party end user enlists on WhatsApp, they keep their user-visible identifier and are also assigned a uniquely generated internal identifier.

**Note:** Details on requirements applicable to identifiers will be specified in the detailed technical documentation.

## 1.1 Verification: Internal Identifiers

Each third party user is represented by an internal identifier that is assigned to them upon enlistment. A third party user is enlisted to a single network, which means the assigned identifier is specific to that network.

When WhatsApp users message third party users, they refer to the internal identifier assigned to the third party at the protocol level (“third party User Identifier”). When third party clients message WhatsApp clients, they refer to the internal identifier assigned to the WhatsApp client.

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

WhatsApp uses the **standard OpenID protocol** (with some minor modifications), alongside the JWT tokens spec, to verify third party identities. When a third party end user attempts to enlist or connect to the chat channel, WhatsApp then verifies their third party 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.

Third party servers are required to generate a signed token verifying the third party user's identity. WhatsApp will periodically fetch the public key from the third party servers. WhatsApp uses this public key to verify the tokens generated by the third party, as follows:

> ***(1)*** *The WhatsApp server periodically fetches the public key through an HTTP endpoint that the third party exposes. This public key is used for signature verification of the token.*
>
> ***(2)*** *The third party server generates a token for the authenticated third party client.*
>
> ***(3)*** *The third party 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 Third party User Enlistment

WhatsApp exposes an Enlistment API that third party 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 third party user identifier is also mentioned in the token.
- Assign an internal identifier to the third party client.
- Check the third party client's 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 3rd party client to request to enlist the third party user on WhatsApp's infrastructure.

After this request successfully completes, the third party user can login to WhatsApp's chat service, also 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).<br><br>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<br>(0x05 == curve25519). |
| e_regid | 4 bytes | The Registration ID. RegistrationID generated by the Signal Protocol |
| e_ident | 32 bytes | e2e identity public key.<br><br>This is the long term key used to create Signal sessions |
| e_skey_id | 3 bytes | e2e skey identifier.<br><br>Also referred to as the Signed PreKey ID, generated by the signal protocol |
| e_skey_val | 32 bytes | e2e skey bytes |

| | | |
|---|---|---|
| | | The public part of the Signed Prekey (a medium term key), generated by the signal protocol |
| e_skey_sig | 64 bytes | skey signature<br><br>The signature over the Signed Prekey by the Identity Key |
| integrator_id | | ID assigned to the third party integrator during onboarding e.g., "1". A base10 string representation (1-999). |
| user_identifier | 1-4 bytes | The third party user's identifier, e.g. "alice" or "16505551234".<br>- Note: Each UTF-8 character is 1-4B.<br>- Restrictions: Only UTF-8 strings of length 1 to 40 are allowed. |
| token | 4 kilobytes | JWT token issued by third party 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=x6g6tGpek0Obx65-vVx9t3wNtjJUXcQY2iarR-
Aush4" \
      --data-urlencode "e_keytype=BQ" \
      --data-urlencode "e_regid=x6g6tA" \
      --data-urlencode "e_skey_id=_tdJ" \
      --data-urlencode
"e_skey_sig=BPEcejlI7RhacXe9MvBMUEs_IHctlcDyza24vDrZca0N77YDGqCAXr2VWDALvOc
TSziNWfVMJ4IDxIENwvyACA" \
      --data-urlencode "e_skey_val=_tdJvCOi0-
F9hl7iaHhrn42d1GnI3p6LwkFx3BS_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 3rd Party Identity Verification

The third party messaging client provides auth token as the identity proof to WhatsApp. WhatsApp then cryptographically verifies if the identity does in fact belong to the third party user in question, through the public keys fetched from the third party 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": "ylVSkbdJm-
Q76nrrPpv7RaGlQVjRfxw5z9OwkM7qzFdUW15QJFrN8nozL21VnOs-
2rAfsoctoA8aQVMmtuWsyeGIpdLrupMVsKqC9bS5_7CQFXRGhQbxsZN1WDU9PcyzQamqFMfOEl3
```

```
Blj3gZr13TMHxaOB0IH6OeMDlzUGWIsbrX4_QyYXr2Wj8xNNdZ7NwmRrik0dFTBg_-
TjTvBzV10WkhbnHT7Lukri1XSEKhjGqAeD0zfGaixhunx3PcFtuB5HNB9qdpkxMeDUG0oUZ3MX-
WY5ppfYMBs8euR1OPG26BAcBM5j4OFuCy8zEv5ornodoa71Af-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 WhatsApp's server, similar to TLS.

## 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 *(detailed stanzas will be sent in the detailed technical documentation):*

- <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:***

```
[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.">>].
```

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

# Part 3. Enabling Messaging

WhatsApp clients use the Signal Protocol to implement end-to-end encryption. Third party clients must also use this protocol in order to send and receive messages with WhatsApp clients.

As mentioned in the Enlistment API's encryption parameters, the third party 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 third party 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 third party account at any given time, hence, WhatsApp clients only encrypt messages to a single device.

Review the <*WhatsApp Security Whitepaper*>, available publicly on 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 third party will need to build message XML stanzas and the content Protobuf structures that are then later encrypted using the Signal Protocol.

More details will be provided in the technical documentation.

## 3.2 Enabling Incoming Messages from WhatsApp

Third party clients must be connected to WhatsApp's servers 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. 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 third party messaging client is offline, WhatsApp servers will store the encrypted messages sent to the user for 30 days in their offline mailbox. In order to facilitate timely delivery of messages, WhatsApp server will notify third party servers over HTTP that their user's client should reconnect to WhatsApp servers and fetch the latest messages waiting for them in their offline mailbox.

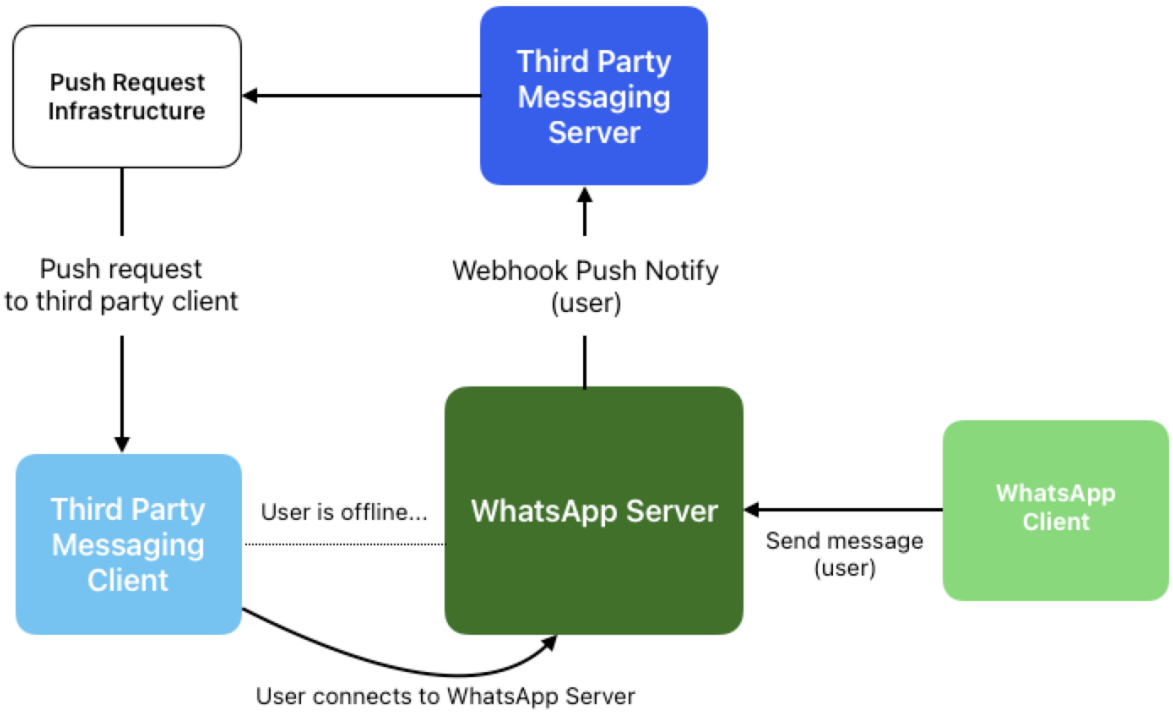



**Figure 1.4:** Push Notification Process Flow

### 3.2.2 Push API Example

Whenever there is new activity for the third party user and the third party client is not connected, WhatsApp makes a HTTPS POST request to the /push path (eg. https://foobar.com/whatsapp/push) with the user identifier as the payload.

***Sample Request***

```
curl --request POST
--url $BASE_URL/push \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer
eyJhbGciOiJSUzI1NiIsImtpZCI6IjFhWGY4IiwidHlwIjoiSldUIn0.eyJpc3MiOiJXSEFUU0F
QUCIsImF1ZCI6Imh0dHBzOi8vaW50ZWdyYXRvci5jb20iLCJpYXQiOjE2ODg2NjkyMzIsImV4cC
I6MTY4ODY3MjgzMn0.lJU_8k9SpbTo-
MdGPdMqWNpdXRlNBwYU3HgkDJmHC9FDg6MQgwEByU7hy-
o2Wo_aO9kf8plJKYJQoxzbGsBfq3uANn4ixKN2TsraV5VxkBtXjhDHRdifYQ1Zk0KlEQ5O1K5oV
0eU_WWjG7ykkb69AsIbw-phF__-
t__zxD8t2fI6TLsqOj3i3zxIBLnR1kuvuHL2DR21JFV74TmAXPXeZQfxLrv542xV7ClJ-
ruBGS2hcbO4M_OVvLbyXBPzP8M6nllhc46OzSTmasTnzxlEtb7pGFsn5eJwZVeWWUQ3mbwDS7aT
aC538PoA_EX8TpLMqxgO5KkTGngKe8gygwcPFg'
--tlsv1.3
--http1.1
--data '{"to": "alice421"}'
```

***Sample Response***

```
200 OK
```

**Note:** There is no payload expected in the response.

The authorization token will be a JWT token signed using WhatsApp server's private key. The third party server should verify it using the public keys downloaded from https://whatsapp.com/.well-known/jwks.

# Part 4. Enabling Additional Message Features and Integrity

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

Optionally, the third party can also choose to implement reporting functionality to give WhatsApp a signal for when WhatsApp users are acting in breach of relevant terms and policies or otherwise causing harm to other users.

## 4.1 Media Management

Third party 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 third party messaging servers using a WhatsApp proxy service.

**Note:** Third party messaging client applications will download media from WhatsApp servers for media messages that are sent by the WhatsApp client application.

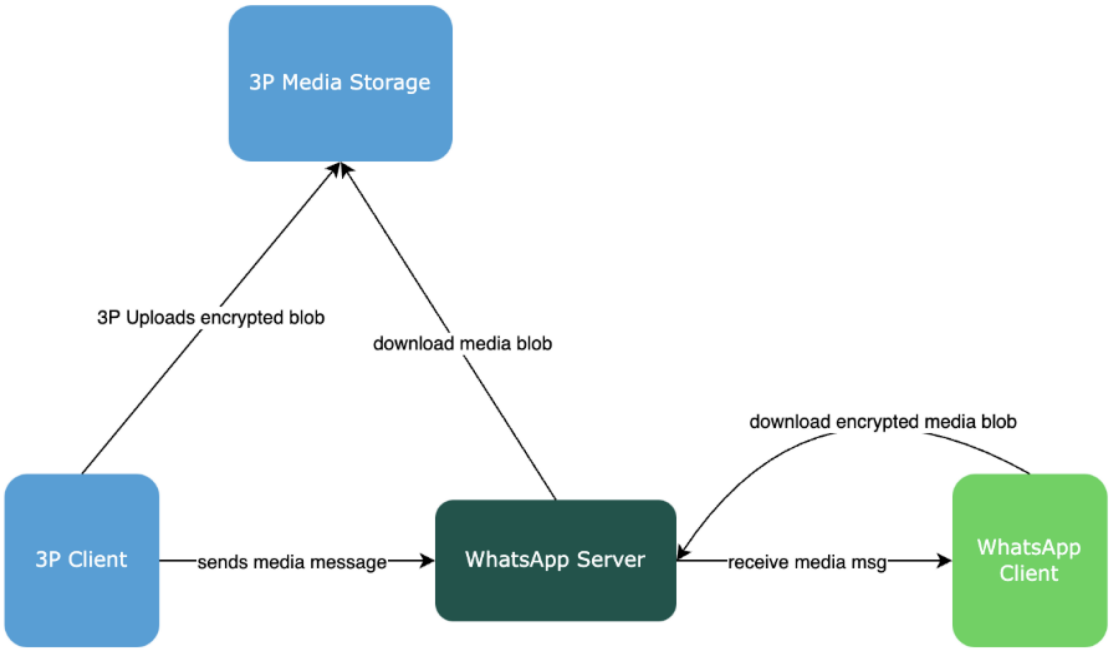



**Figure 1.3:** Media Management Process Flow

### 4.1.1 Media API Example

After the WhatsApp client uploads media to the upload service, it will get back a direct path (i.e., /v/t{file_identifier}) from the upload service response. The WhatsApp client will then pass the direct path inside the media message.

The third party client should construct the full download URL by prepending the host base_url for download (e.g. http://mmg.whatsapp.net/v/t{file_reference}). The third party 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 & Reporting

WhatsApp ensures that third party clients never receive messages from a WhatsApp user that they've blocked.

To block a WhatsApp user or to read the current list of users that the third party client has blocked, getting and setting blocklist IQs are used over the chat channel as described in Part 2.1.

# Part 5. User Deletion

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

If a third party client does not connect to WhatsApp servers for a period of 30 days, WhatsApp will automatically remove their details from the WhatsApp servers. If this occurs, the third party clients will receive an error, and must re-enlist as described previously.

# Part 6. Updates and Ongoing Support

WhatsApp will communicate any changes, updates, issues, bugs, etc. to qualifying third parties through a dedicated communication channel. Any updates that require changes from third party developers will be communicated in advance.

Third party developers will be provided access to a WhatsApp point of contact for ongoing communication and support.

### 6.1 Testing

Qualifying third parties will receive detailed implementation instructions broken down into milestones with testable experiences. The third party can communicate with their WhatsApp point of contact for assistance while completing each milestone.

**Note:** Support program details for initial third party messaging partners are currently under review and subject to change.

### 6.2 Service Level Agreements

Each party will use best endeavors to maintain service and communication standards.

### 6.3 Versioning

The WhatsApp protocol supports versioning. Regular updates to the protocol will be published and documented, and third party messaging platforms will have a reasonable notice period to adopt the latest supported version.

Third party messaging clients are required to provide the protocol version they are using when connecting to WhatsApp servers.

A version changelog will be provided and published regularly for partners to review. The changelog will list any modifications that have been made to processes, endpoints, and other technical implementation details (such as protocol enhancements).

# Glossary

**3P -** Shorthand for ‘third party’ messaging service. You may see this in the Developer Guide and in some of the API endpoint examples and diagrams.

**Chat channel -** The communications channel established and maintained between WhatsApp and third party messaging services. The channel is between the client and the server (chatd).

**Devices -** End user mobile devices (iOS and Android) that are used when sending and receiving messages between WhatsApp and third party 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 WhatsApp messaging service (servers, API endpoints, etc.).

**Internal identifier -** A unique identifier for the third party messaging service (created and used by WhatsApp). Used at the protocol layer.

**Third Party User Identifier -** A unique identifier for the third party messaging service end user (phone number, email, etc.). Displayed to the end users.

**User Enlistment -** The process of setting up a new third party end user on WhatsApp.

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

**WhatsApp Registration Server -** The WhatsApp server that manages the client application and third party messaging user registration 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.