The WhatsApp Multi-Device History Synchronization Protocol

1. Introduction

The WhatsApp Multi-Device History Synchronization Protocol enables a newly linked companion device to receive a user's recent chat history from their primary device. The protocol is designed to transfer a large amount of data efficiently and securely.

History is not sent message-by-message. Instead, it is bundled into a large, encrypted, and compressed blob, which is uploaded to a WhatsApp server and subsequently downloaded by the companion device. The entire process is end-to-end encrypted, ensuring the server cannot access the contents of the user's chat history.

2. Overview

2.1. Terminology

- **Primary Device**: The user's main mobile phone, which is the source of the historical data.
- **Companion Device**: A newly linked device that needs to receive the chat history.
- **History Sync**: The overall process of transferring chat history.
- **Sync Blob**: A large, encrypted, and compressed file containing the serialized history data.
- **Media Key**: A 32-byte secret key used to derive the keys for decrypting the Sync Blob.

2.2. Cryptographic Primitives

- **HKDF**: HMAC-based Key Derivation Function using SHA-256.
- **AES-256-CBC**: Used for the encryption and decryption of the Sync Blob.
- **HMAC-SHA256**: Used to verify the integrity of the encrypted Sync Blob.
- **zlib**: Used for compression and decompression of the history payload.

3. Protocol Flow

The history sync process is initiated by the primary device after a new companion device has been successfully linked.

3.1. Step 1: History Sync Notification

1. The primary device bundles a portion of the user's recent chat history into a HistorySync Protobuf message.

2. This payload is compressed using zlib.
3. The compressed data is then encrypted to create the Sync Blob (see Section 4 for details).
4. The primary device uploads this Sync Blob to a WhatsApp media server, receiving a directPath and other metadata in return.
5. The primary device sends a standard end-to-end encrypted message to the companion device. This message contains a HistorySyncNotification Protobuf, which includes the download metadata (directPath, fileEncSha256, etc.) and the mediaKey required for decryption.

3.2. Step 2: Media Download

1. The companion device receives the HistorySyncNotification.
2. It uses the directPath and other metadata to download the encrypted Sync Blob from the WhatsApp media server. This is a standard media download operation.

3.3. Step 3: Decryption and Decompression

1. Using the mediaKey from the notification, the companion device derives the necessary decryption and MAC keys (see Section 4.1).
2. It verifies the integrity of the downloaded Sync Blob using the derived MAC key and then decrypts it using the derived cipher key.
3. The resulting plaintext is the zlib-compressed history payload. The companion decompresses this data to retrieve the original HistorySync Protobuf message.

3.4. Step 4: Processing the Payload

The companion device processes the contents of the HistorySync message, populating its local database with conversations, messages, contact names, and other settings. The protocol may involve multiple rounds of notifications and downloads to transfer the complete history in chunks.

4. Key Derivation and Decryption

The security of the history sync relies on the end-to-end encryption of the Sync Blob.

4.1. Media Key Expansion

The 32-byte mediaKey received in the HistorySyncNotification is expanded into an 80-byte key using HKDF-SHA256. The info parameter for the HKDF is the ASCII string "WhatsApp History Keys".

ExpandedKey = HKDF-SHA256(salt=nil, inputKeyMaterial=mediaKey, info="WhatsApp History Keys", outputLength=80)

The resulting 80 bytes are split into:

- **IV** (ExpandedKey[0:16]): A 16-byte initialization vector for AES-CBC.
- **Cipher Key** (ExpandedKey[16:48]): A 32-byte (256-bit) key for AES-CBC.
- **MAC Key** (ExpandedKey[48:80]): A 32-byte key for the HMAC-SHA256 integrity check.

4.2. CBC Decryption and Validation

The downloaded Sync Blob consists of the AES-CBC ciphertext followed by a 10-byte truncated HMAC-SHA256 of the ciphertext and IV.

1. **Integrity Check**: The companion computes HMAC-SHA256(key=MACKey, data=IV || Ciphertext) and verifies that the first 10 bytes match the MAC appended to the downloaded blob.
2. **Decryption**: If the MAC is valid, the companion decrypts the ciphertext using AES-256-CBC(key=CipherKey, iv=IV, data=Ciphertext).
3. **Unpadding**: The decrypted data is unpadded according to standard PKCS#7 rules. The result is the compressed HistorySync payload.

5. Data Structures

5.1. HistorySyncNotification

This is a Protobuf message sent from the primary to the companion device to initiate a sync.

- **fileSha256**: The SHA-256 hash of the original, unencrypted, uncompressed HistorySync payload.
- **fileLength**: The length in bytes of the original payload.
- **mediaKey**: The 32-byte secret key for decryption.
- **fileEncSha256**: The SHA-256 hash of the final encrypted Sync Blob.
- **directPath**: The URL path for downloading the blob from the media server.
- **syncType**: An enum indicating the type of history being synced (e.g., initial bootstrap, recent, full).

5.2. HistorySync Payload

This is the main Protobuf message contained within the Sync Blob. It can contain a variety of data types.

- **syncType**: The type of sync.
- **conversations**: A list of Conversation objects, each containing chat metadata and a list of Message objects.
- **pushnames**: A list of Pushname objects, mapping user JIDs to their display names.

- **phoneNumberToLidMappings**: A list of mappings between Phone Number JIDs and LID JIDs.
- **globalSettings**: A GlobalSettings object containing account-wide settings.

6. Payload Content Types

The HistorySync payload is a container for different kinds of historical data.

6.1. Conversations and Messages

This is the primary content. Each Conversation object includes the chat JID and metadata. The contained Message objects are standard WebMessageInfo Protobufs, which include the message content, sender, timestamp, and a messageSecret for use in other protocols (like Poll Votes or Reactions).

6.2. Push Names

To populate the companion device's contact list, the primary device sends a list of known push names (display names) associated with user JIDs.

6.3. Phone Number to LID Mappings

For accounts that have undergone the LID (Lightweight Identity) migration, this provides a mapping between a contact's stable phone number JID and their new, privacy-preserving LID JID.

6.4. Global Settings

This can include various account-level settings, such as the timestamp of the account's migration to the LID system.

7. Security Considerations

- **End-to-End Encryption**: The history data is encrypted with a key that is only ever shared between the user's own devices over an end-to-end encrypted channel. The server only stores an opaque blob and has no ability to decrypt it.
- **Integrity**: The use of HMAC-SHA256 ensures that the Sync Blob cannot be tampered with in transit without detection.
- **Ephemeral Nature**: The directPath for the Sync Blob is typically short-lived, reducing the window of opportunity for an attacker to attempt to download it.
- **Source Authentication**: The HistorySyncNotification is sent within a standard, authenticated Signal Protocol session between the primary and companion devices, ensuring the notification and the mediaKey are from the legitimate primary device.

8. References

This document is based on analysis of the open-source whatsmeow library, an unofficial Go implementation of the WhatsApp Web API.