Technical Specification: WhatsApp Pre-Keys and X3DH

1. Introduction

1.1. Purpose

This document provides a detailed, language-agnostic specification for the pre-key system used by WhatsApp to establish end-to-end encrypted (E2EE) sessions. It is based on the Extended Triple Diffie-Hellman (X3DH) key agreement protocol.

1.2. Scope

The specification covers the types of pre-keys, their lifecycle, and the protocol flow for using them to create a secure, asynchronous communication channel. It is intended as a foundational guide for developers implementing the session establishment phase of the WhatsApp protocol.

1.3. Terminology

- **Pre-Key:** A public key published to a server in advance by a user, allowing others to initiate an E2EE session with them even if they are offline.
- **IK (Identity Key):** A long-term static key pair that represents a user's identity.
- **SPK (Signed Pre-Key):** A medium-term key pair that is signed by the Identity Key and periodically replaced.
- **OPK (One-Time Pre-Key):** A single-use key pair used to provide forward secrecy.
- **Pre-Key Bundle:** A collection of a user's public keys (IK, SPK, and one OPK) fetched by an initiator to start a new session.
- **X3DH:** Extended Triple Diffie-Hellman, the key agreement protocol that uses pre-keys.

2. The Role of Pre-Keys in Asynchronous E2EE

The primary challenge in establishing an E2EE session is that it typically requires an interactive key exchange where both parties are online simultaneously. The pre-key model solves this problem for asynchronous communication (e.g., sending a message to a user who is offline).

The core idea is that a user (let's call him Bob) pre-emptively generates a set of public keys and uploads them to the WhatsApp server. When another user (Alice) wants to send an initial message to Bob, she fetches this "pre-key bundle" from the server. Using the bundle, Alice can perform a series of Diffie-Hellman (DH) calculations to derive a shared secret key, encrypt her first message, and send it to Bob. When Bob comes online, he can use his corresponding private keys to derive the same shared secret and decrypt the message.

3. Types of Pre-Keys

The X3DH protocol utilizes three distinct types of keys to achieve mutual authentication and forward secrecy.

3.1. Identity Key (IK)

The Identity Key is a long-term Curve25519 key pair that forms the foundation of a user's cryptographic identity. It is relatively static and is used to sign the Signed Pre-Key, proving Bob's ownership of it.

3.2. Signed Pre-Key (SPK)

The Signed Pre-Key is a medium-term Curve25519 key pair.

- **Lifecycle:** It is generated by the client and replaced periodically (e.g., weekly or monthly).
- **Function:** It serves as a primary, authenticated touchpoint for key agreement.
- **Signature:** To prevent tampering by the server, the public part of the SPK is signed by the private part of the Identity Key using the XEdDSA scheme. This signature is also uploaded to the server. An initiator (Alice) must verify this signature before using the SPK.

3.3. One-Time Pre-Keys (OPKs)

One-Time Pre-Keys are a batch of Curve25519 key pairs generated and uploaded by a client.

- **Lifecycle:** As their name implies, each OPK is intended for a single use. When the server delivers an OPK to an initiator as part of a pre-key bundle, it immediately deletes it from its storage.
- **Function:** The use of an OPK provides **forward secrecy** for the initial session. Once a session is established using an OPK and that key is deleted, even a full compromise of Bob's other keys (IK and SPK) at a later date cannot be used to retroactively decrypt the initial message.
- **Management:** Clients maintain a pool of OPKs on the server. Implementations (like whatsmeow) should periodically check the number of available keys on the server. If the count drops below a minimum threshold (e.g., 5), the client should generate and upload a new batch to reach a target count (e.g., 50).

4. The Pre-Key Bundle

When Alice wants to start a session with Bob, she requests his pre-key bundle from the server. This bundle contains:

1. **Bob's Identity Key Public Key** (IK_B.pub)
2. **Bob's Signed Pre-Key Public Key** (SPK_B.pub)
3. **The signature on the SPK** (Sig(IK_B.priv, SPK_B.pub))
4. **(Optionally) One of Bob's One-Time Pre-Key Public Keys** (OPK_B.pub)

If the server has run out of OPKs for Bob, the bundle will not contain one. The protocol gracefully handles this scenario, albeit with slightly different security properties for the initial message.

5. Protocol Flow: Establishing a Session (X3DH)

The process of establishing a session can be broken down into the following steps:

5.1. Step 1: Publishing (Bob)

Bob's client connects, generates its keys, and publishes the following to the server via an IQ stanza:

- His public Identity Key (IK_B.pub).
- His public Signed Pre-Key (SPK_B.pub) and its signature.
- A batch of public One-Time Pre-Keys (OPK_B.pub_1, OPK_B.pub_2, ...).

5.2. Step 2: Fetching (Alice)

Alice's client requests the pre-key bundle for Bob from the server, specifying Bob's JID.

5.3. Step 3: Key Agreement (Alice)

1. Alice generates her own long-term Identity Key pair (IK_A) and a fresh **ephemeral** key pair (EK_A).
2. She verifies the signature on Bob's SPK_B using IK_B.pub. If it fails, she aborts.
3. She performs a series of DH calculations to compute a shared secret (SK).
   - DH1 = DH(IK_A.priv, SPK_B.pub)
   - DH2 = DH(EK_A.priv, IK_B.pub)
   - DH3 = DH(EK_A.priv, SPK_B.pub)
   - **If an OPK is present:** DH4 = DH(EK_A.priv, OPK_B.pub)
4. She combines the results using a Key Derivation Function (KDF), typically HKDF. The inputs are concatenated in a fixed order.
   - **With OPK:** SK = KDF(DH1 || DH2 || DH3 || DH4)
   - **Without OPK:** SK = KDF(DH1 || DH2 || DH3)

5.4. Step 4: Initial Message (Alice)

Alice sends an initial message to the server for Bob. This is an <enc> (encrypted) message stanza which includes:

- Her public Identity Key (IK_A.pub).
- Her public Ephemeral Key (EK_A.pub).
- The identifier of the pre-keys she used from Bob.
- An encrypted payload containing her first message. This payload is encrypted as a "pre-key message" (pkmsg) using a key derived from SK.

5.5. Step 5: Session Creation (Bob)

1. When Bob comes online, he receives Alice's initial message from the server.
2. He uses his private keys (IK_B.priv, SPK_B.priv, and the relevant OPK_B.priv) and Alice's public keys from the message (IK_A.pub, EK_A.pub) to perform the exact same DH and KDF calculations, deriving the same shared secret SK.
3. He uses SK to derive the message key and decrypt the payload.
4. Crucially, he immediately **deletes the private key for the OPK** that was used. This ensures forward secrecy for this session.
5. With the session established, future messages between Alice and Bob will be managed by the Double Ratchet algorithm, which evolves the session keys with each message.

6. Pre-Key Management (Client-Side)

A robust client implementation is responsible for managing the lifecycle of its own pre-keys.

- **Generation:** All keys must be generated using a cryptographically secure random number generator. Keys are Curve25519 pairs.
- **Storage:**
    - The **Signed Pre-Key** (including its ID and signature) is typically stored with the main device/identity information (e.g., in the whatsmeow_device table).
    - The pool of **One-Time Pre-Keys** must be stored separately (e.g., in the whatsmeow_pre_keys table). The storage should track each key's ID, the private key, and a flag indicating if it has been successfully uploaded to the server.
- **Uploading & Maintenance:** The client must periodically ensure a sufficient number of OPKs are available on the server.
    1. Send an <iq> stanza to the server to query the current OPK count.
    2. If the count is below a minimum threshold (e.g., 5), generate a new batch of keys to bring the total up to a desired level (e.g., 50).
    3. Upload the new public keys in a single <iq> stanza.
    4. Upon successful acknowledgment from the server, mark the corresponding keys as "uploaded" in the local database.

7. Security Considerations

- **Forward Secrecy:** The use and subsequent deletion of One-Time Pre-Keys are essential for providing forward secrecy. If an attacker compromises Bob's keys *after* a session is established, they cannot decrypt past messages that were initiated with an OPK.
- **Pre-Key Depletion:** If a user is very popular, their OPKs may be depleted. In this case, the protocol falls back to a Triple DH agreement (without DH4). The initial message has weaker forward secrecy guarantees. Its security relies on the medium-term SPK. Security is restored once the Double Ratchet algorithm kicks in after the first reply from Bob.
- **Server Trust:** The server is treated as an untrusted intermediary. It stores public keys but cannot derive the shared secret SK because it does not possess any of the private keys involved in the DH calculations. Its primary role is to honestly provide pre-key bundles

and route the initial message. Tampering with the bundle is prevented by the SPK signature.

- **Authentication:** X3DH provides mutual cryptographic authentication if both parties trust the identity keys they are using. This trust is typically established out-of-band (e.g., by scanning a QR code with a safety number or verifying it manually). Without this step, parties are vulnerable to man-in-the-middle attacks where they establish a secure