Arattai

Encryption Overview

White Paper

04 June 2026

INDEX OF SECTIONS

End-to-end encryption (E2EE) ensures that only the sender and intended recipient can read messages, protecting data from ISPs, service providers, hackers, and other intermediaries. Currently, Arattai supports E2EE for direct chats on Android (version 1.33.6 or above), iOS (version 1.17.23 or above), and the Arattai Web and Desktop apps (version 1.0.7 or above).

Cryptographic Foundations

Arattai's E2EE is built on the **Signal protocol**, leveraging open-source cryptographic libraries. The protocol uses the Diffie-Hellman key exchange to securely establish shared secrets over insecure channels, ensuring that session keys are never transmitted directly. Arattai provides multi-device end-to-end encrypted chats so users can seamlessly send messages from any of their linked devices. A user can have a maximum of 5 linked devices.

Key Generation

To initiate an E2EE chat, both the initiator and the recipient must exchange cryptographic key pairs to establish secure, encrypted sessions with each of the recipient's devices. The initiator's device contacts the server and requests a pre-key bundle for all of the recipient's devices, as well as any other linked devices belonging to the initiator.

- **Identity Key** is a long-term asymmetric key pair generated once per user when they sign in on a new device. It remains valid for the lifetime of the app installation and active login session on that device. The Identity Key is generated using Curve25519 and serves as the root of trust for authenticating other keys.
- **Signed Pre-Key** is a semi-long-term Curve25519 key pair whose public key is authenticated by a digital signature created using the user's Identity private key (XEdDSA signature). The Public Signed Pre-Key and its signature are uploaded to the server as part of the user's key bundle. To limit the exposure window in case the Signed Pre-Key is compromised, it is rotated periodically, typically every 15 days.
- **One-Time Pre-Keys** are short-lived, single-use Curve25519 key pairs designed to provide forward secrecy during session establishment. A client generates a batch of One-Time Pre-Keys (for example, ~100 keys) and uploads their public components to the server during registration. Each One-Time Pre-Key is

consumed at most once and deleted after use. Clients periodically replenish the server's supply when it drops below a defined threshold.

- **Ephemeral Keys** are transient Curve25519 key pairs generated for each session initiation. They are used only during the key agreement phase and are discarded immediately after the session keys are derived, ensuring strong forward secrecy.

Device Registration

When a user registers by signing up or when a user logs in to a new device, three key pairs, namely Identity Key (IK), Signed Pre-Key (SPK), and a set of One-Time Pre-Keys (OPK1, OPK2, OPK3, ...), are generated. The server stores these public keys along with device identifiers generated by the Signal protocol. The private keys are stored on the device after the registration process is successful.

Establishing an Encrypted Session

When a user initiates an E2EE chat with another user, the initiating device sends a request to the server to obtain the recipient's pre-key bundle. The server responds by delivering the corresponding pre-key bundle to the requesting device. Each device's pre-key bundle comprises the following elements:

- Recipient's Public Identity Key => $IK_{recipient}$
- Recipient's Public Signed Pre-Key => $SPK_{recipient}$
- Recipient's Pre-Key Signature => $Sig(IK_{recipient}, Encode(SPK_{recipient}))$
- Recipient's Public One-Time Pre-Key => $OPK_{recipient}$

Note that each pre-key bundle consists of only a single One-Time Pre-Key from a set of the recipient's One-Time Pre-Keys. The provided One-Time Pre-Key is deleted from the server as soon as it is sent to the user, as it should be used only for a single session. Once the pre-key bundle is received from the server, the initiator verifies the pre-key signature and aborts the protocol if verification fails.

X3DH

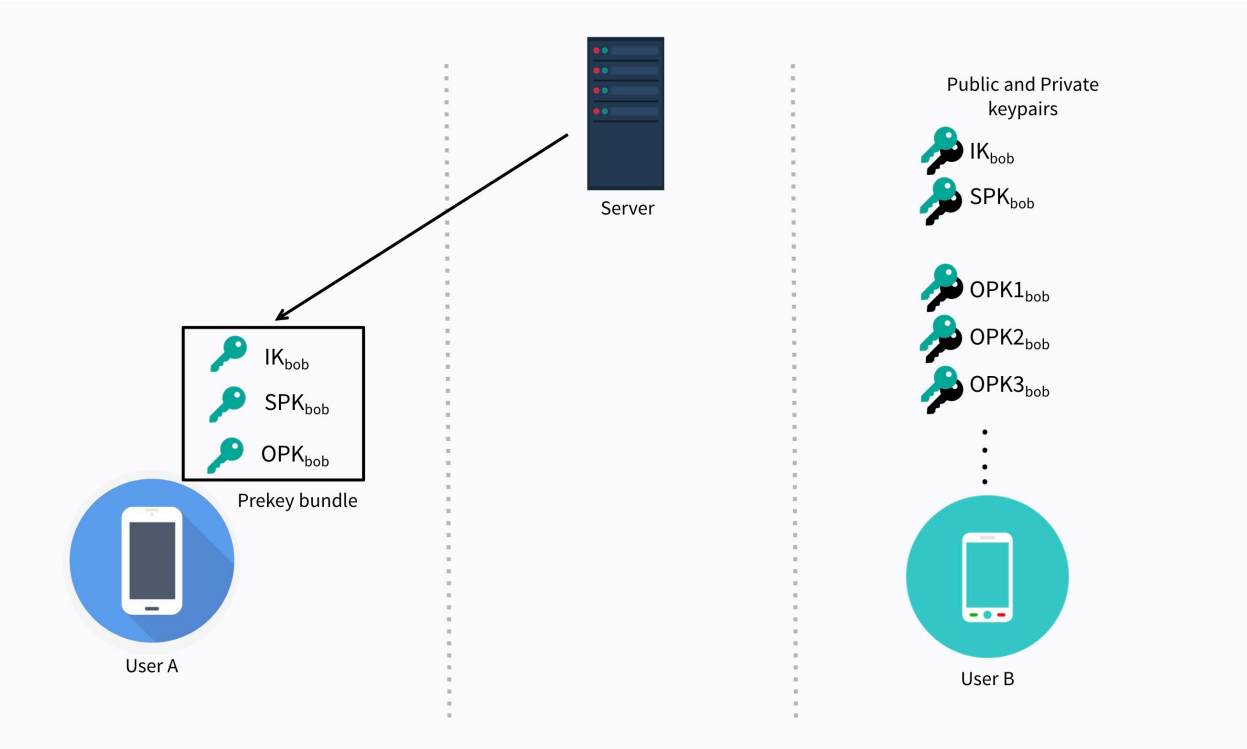
Extended Triple Diffie-Hellman (X3DH) is a key agreement protocol utilised by the Signal protocol to establish a shared secret between two users, even when one party is offline. It performs multiple Diffie-Hellman (DH) exchanges involving long-term, signed, one-time, and Ephemeral Key pairs to ensure authentication, forward secrecy, and deniability. Upon successful execution, X3DH outputs a shared secret key (SK) derived through the following computations:

- DH1 = DH ($IK_{initiator}$, $SPK_{recipient}$)
- DH2 = DH ($EK_{initiator}$, $IK_{recipient}$)
- DH3 = DH ($EK_{initiator}$, $SPK_{recipient}$)
- DH4 = DH ($EK_{initiator}$, $OPK_{recipient}$)
- SK = KDF (DH1 || DH2 || DH3 || DH4)

$EK_{initiator}$ refers to the Ephemeral Key generated by the initiator at the time of session creation to provide forward secrecy. The Diffie-Hellman outputs are concatenated and passed to a Key Derivation Function (KDF), which generates the final shared secret key SK. After deriving SK, the initiator permanently deletes the ephemeral private key and all intermediate DH outputs to maintain forward secrecy.

At this stage, a secure session is established on the initiator's side, allowing encrypted messages to be sent even if the recipient is offline. The recipient becomes aware of the initiator's public keys upon receiving and decrypting the first encrypted message, completing the session setup on their end.


Public keys
IK$_{bob}$
SPK$_{bob}$
Server
OPK1$_{bob}$
OPK2$_{bob}$
OPKn$_{bob}$
Public and Private key pairs
IK$_{bob}$
SPK$_{bob}$
OPK1$_{bob}$
OPK2$_{bob}$
OPK3$_{bob}$
User A
User B


Server
Public and Private keypairs
IK$_{bob}$
SPK$_{bob}$
OPK1$_{bob}$
OPK2$_{bob}$
OPK3$_{bob}$
IK$_{bob}$
SPK$_{bob}$
OPK$_{bob}$
Prekey bundle
User A
User B

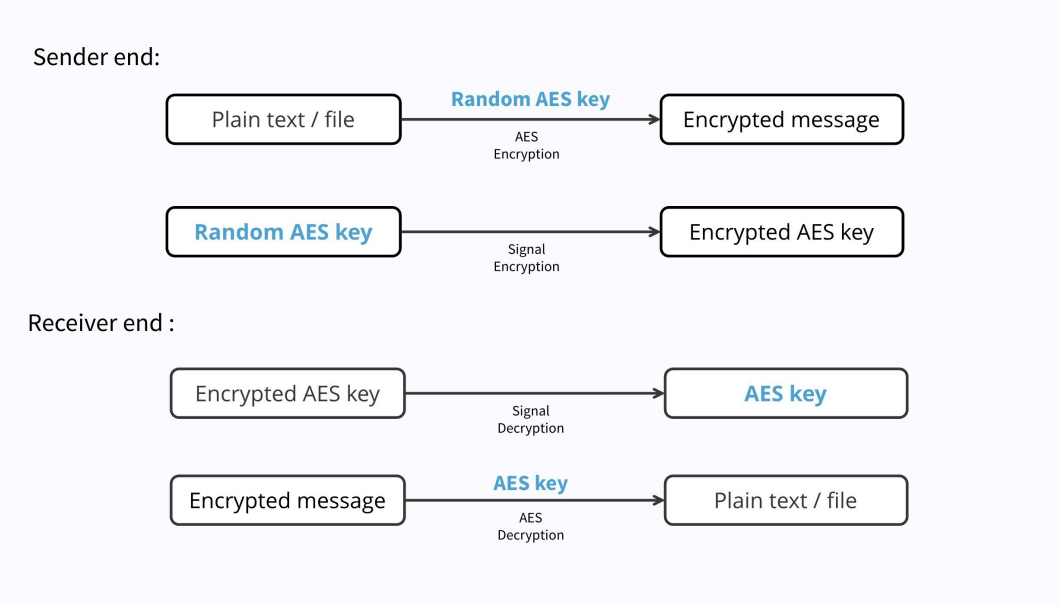
AES-GCM

Arattai utilises the Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) for message content encryption. AES is a symmetric encryption algorithm widely recognised for its security and efficiency, making it an ideal choice for protecting sensitive information in real-time messaging environments.

The implementation employs a random **256-bit key** for each message, providing a strong defence against brute force attacks while maintaining performance. This key length represents the current industry standard for securing sensitive data, offering a balance between security and processing efficiency.

For each encryption operation, a random **12-byte initialisation vector (IV)** is generated for each message. This ensures that even when encrypting identical messages, the resulting ciphertexts will be different. The randomisation prevents pattern analysis and replay attacks, adding an important layer of security to the encryption scheme.

The **Galois/Counter Mode (GCM)** adds authentication capabilities to the encryption process. Unlike basic encryption modes that only provide confidentiality, GCM simultaneously verifies the integrity of the encrypted data. This means the system can detect if a message has been tampered with during transmission, protecting users from man-in-the-middle attacks.



During encryption, the plaintext message or file data is processed using the AES-GCM algorithm with the generated key and IV. The resulting ciphertext is paired with its corresponding IV for transmission. The AES key used for encrypting the message is encrypted using Signal protocol for each available device of the users in the chat. When receiving an encrypted message, the Signal-encrypted AES key is first decrypted. The AES decryption process extracts the IV from the received data. It then applies the same AES key with the extracted IV to convert the ciphertext back into the original plaintext. This symmetry in encryption and decryption operations allows for efficient processing on both sending and receiving devices.

The system ensures forward secrecy through the Signal protocol's Double Ratchet mechanism, which continuously derives new encryption keys for every message. Even if a current key is compromised, previously exchanged messages remain secure because earlier keys cannot be reconstructed. This creates an independent security boundary around each message and minimises the impact of any potential breach.

Text Message & File Encryption

Symmetric Encryption Protocol

```
Algorithm : AES-GCM
Key Length : 256 bits
IV Length : 12 bytes (random)
Authentication Tag: 128 bits
```

Message Format

```
EncryptedMessage = Base64(IV) + "$" + Base64(Ciphertext)
```

Signal Protocol Key Exchange

```
DH1 = DH(IKinitiator, SPKrecipient)

DH2 = DH(EKinitiator, IKrecipient)

DH3 = DH(EKinitiator, SPKrecipient)

DH4 = DH(EKinitiator, OPKrecipient)

SK = KDF(DH1 || DH2 || DH3 || DH4)
```

Message Encryption Process

```
AESKey = Random(256 bits)

IV = Random(12 bytes)

Ciphertext = AES-GCM-Encrypt(AESKey, IV, PlainText)

EncryptedAESKey = Signal_Encrypt(SessionKey, AESKey)

TransmittedMessage = {EncryptedAESKey, Base64(IV) + "$" + Base64(Ciphertext)}
```

Message Decryption Process

```
AESKey = Signal_Decrypt(SessionKey, EncryptedAESKey)

[IV_Base64, Ciphertext_Base64] = Split(EncryptedData, "$")

IV = Base64Decode(IV_Base64)

Ciphertext = Base64Decode(Ciphertext_Base64)

PlainText = AES-GCM-Decrypt(AESKey, IV, Ciphertext)
```

Forward Secrecy and Double Ratchet

The Signal protocol ensures robust end-to-end encryption through two key mechanisms: Forward secrecy and the Double Ratchet algorithm.

Forward secrecy guarantees that the compromise of long-term keys does not expose previously exchanged messages. Each message is encrypted with a unique, one-time key derived from evolving key chains. Once used, these keys are deleted, preventing any decryption of past communication.

The Double Ratchet algorithm enhances this security by continuously updating encryption keys as messages are exchanged. It combines:

- A Diffie-Hellman ratchet that introduces new randomness whenever new key pairs are exchanged, and
- A Symmetric-key ratchet that derives a fresh key for every message using a key derivation function (KDF).

Together, these ratchets provide both forward secrecy and post-compromise security, ensuring that even if a device is compromised, past messages remain protected and future messages quickly regain full confidentiality.

New Device Addition and Old Device Removal

Once an end-to-end encrypted chat is established and communication is ongoing, users may add a new linked device or remove an existing one. Such actions introduce potential device conflicts that must be resolved to maintain end-to-end encryption integrity.

When a device is deregistered, it is immediately removed from the user's list of active linked devices. This ensures that no further encrypted messages are routed to that device, preventing data leakage or unauthorised message access.

Conversely, when a user registers a new device, the new device generates and uploads a fresh pre-key bundle to the server. This bundle is shared with other participants in subsequent message exchanges, allowing the creation of new Signal

sessions for the added device. These sessions enable the device to receive and decrypt future messages securely.

If a user attempts to send a message during this transition period, the server returns a conflict response, signalling the client to fetch the latest pre-key bundles, establish new sessions, and retransmit the message. This mechanism ensures that all active devices remain synchronised and capable of secure message exchange under the updated device set.

Key Replenishment and Forward Secrecy Considerations

To maintain secure session establishment, each device periodically uploads a batch of One-Time Pre-Keys (OTPKs) to the server. These pre-keys are consumed whenever a new session is initiated with that device and are deleted from the server after use.

When the available OTPKs are depleted, the Arattai server notifies the device to replenish the pre-keys. If the device is offline and keys are not replenished, session setup continues using only the Identity Key (IK) and Signed Pre-Key (SPK). In this case, forward secrecy is not achieved during the initial session creation, as the session key can be recomputed if long-term keys are later compromised.

However, once the Double Ratchet algorithm begins operating after session establishment, forward secrecy is restored for all subsequent messages, since each Message Key is derived uniquely and old keys are permanently discarded. To mitigate this limitation, Arattai periodically rotates the Signed Pre-Key (SPK), reducing the window during which a compromised SPK could affect session security.

Server Roles and Limitations

Server-Side Key Storage

The server maintains a minimal set of data to facilitate secure communications while preserving end-to-end encryption principles. It stores device identifiers that

uniquely recognise each client device in the system, timestamps recording when devices were linked to user accounts, and the public components of cryptographic keys used by the Signal protocol.

Key Distribution

The server stores and delivers users' pre-key bundles, including the Public Identity Key, Public Signed Pre-Key, and Public One-Time Pre-Keys, to initiators who intend to establish a secure session.

Message Relay

It functions purely as a transport mechanism, forwarding encrypted messages between users without the ability to inspect or modify their contents.

User Registration and Authentication

The server manages user registration, assigns unique user identifiers, and maintains the mapping between users and their corresponding cryptographic keys.

Offline Message Handling

To support asynchronous messaging, the server stores encrypted E2EE messages.

No Access to Private or Session Keys

The server never possesses private keys or session keys derived through X3DH. All cryptographic computations occur solely on user devices, ensuring end-to-end confidentiality.

Minimal Trust Requirement

The system operates under a minimal trust model, where even a compromised or malicious server cannot decrypt message content due to the properties of end-to-end encryption.

One-Time Pre-Key Management

One-Time Pre-Keys are single-use keys. After a One-Time Pre-Key is fetched from the server as part of session establishment, the server permanently deletes it to prevent subsequent retrievals. The generating client also removes its corresponding private

key once the session has been established using that key. This ensures that each One-Time Pre-Key contributes to only one session setup, preserving forward secrecy.

Metadata Visibility

Although message payloads remain encrypted, the server inherently has access to limited metadata, including sender and recipient identifiers, message timestamps, message types, and reactions. This metadata is protected with EAR (Encryption at Rest) on the server.

E2EE for Direct Messages

Chat types, including group chats, channels, and Pocket, are cloud-based. These are securely stored on the server but not end-to-end encrypted. E2EE chat searches run only on your device, keeping message content private. Cloud-backed chats, on the other hand, allow server-side search.

E2EE Message Backup

End-to-end encrypted backups are not supported at this time for E2EE chats. Support for E2EE backups is planned and will be introduced in a future update.

Broadcast Messages

Broadcast message encryption operates in two phases. In the first phase, the sender's device generates a **Sender Key** and distributes it to every recipient device through the individually established pairwise encrypted sessions described in the **"Establishing an Encrypted Session"** section. In the second phase, every message is encrypted once using that Sender Key, and the server delivers it to all recipients. The per-recipient encryption work is confined to key distribution, not repeated on every message sent.

The Sender Key is generated by the sender's device and is tied to that device and that broadcast. Each recipient device receives an encrypted copy of it solely to verify and decrypt incoming messages; the key cannot be used to originate messages on behalf of the sender.

Sender Key Distribution: First Message

1. The sender's device generates a random 32-byte Chain Key and a Curve25519 Signature Key pair.

2. The Chain Key and the Signature Key's public key are packaged into a **Sender Key Distribution Message (SKDM)**

3. The SKDM is encrypted separately for every registered device of every recipient, each using the pairwise encrypted session already in place with that device. This is **client fan-out**: one encrypted copy per device, isolated from all others.

4. The server delivers each device's copy into that recipient's 1:1 chat, alongside the encrypted message.

Message Encryption: Subsequent Messages

1. The sender's device derives a Message Key from the Chain Key and advances the Chain Key to its next state.

2. The message is encrypted using AES-256 in CBC mode with that Message Key.

3. The ciphertext is signed using the Signature Key.

4. The single ciphertext is submitted to the server, which performs **server-side fan-out**: the same payload is delivered into every recipient's 1:1 chat. Each recipient decrypts independently using the Sender Key already in their possession.

Every Chain Key advancement is a one-way HMAC derivation; the key moves forward, and the prior state is permanently discarded. This is the **hash ratchet**, and it is what provides **forward secrecy**: no prior message can be decrypted even if the current key state is exposed. Every ciphertext is signed with the Signature Key, so recipients can confirm that what they are decrypting was produced by the legitimate sender and has not been altered. Any modification in transit fails verification before decryption is attempted.

Each message carries an iteration counter tied to the Sender Chain Key state. Since the Chain Key is a one-way ratchet, each derived Sender Message Key corresponds

to a unique counter value. If a previously delivered message is re-sent by an attacker, the recipient's device will detect that the counter has already been seen/processed and **reject it as a replay**.

Out-of-Order Message Decryption: Since the Sender Chain Key ratchets forward linearly, out-of-order delivery is handled by caching skipped Sender Message Keys. When a message with an ahead counter arrives, the recipient's device advances the Chain Key forward, storing the intermediate Sender Message Keys for the missing messages. When the out-of-order messages eventually arrive, the cached keys are used to decrypt them.

Sender Key Lifecycle: Ratcheting and Reset

The Sender Key is not static for the lifetime of a Broadcast List. The Chain Key ratchets forward with every message, and the entire Sender Key is reset when membership changes or the key reaches its maximum age. The following events govern when the key is extended to new recipients and when it is fully replaced.

Addition: Recipient or Device

When a new recipient is added or an existing recipient links a new device, the SKDM is distributed to the newly identified devices on the next send via their established pairwise sessions. No Sender Key reset is required. However, since the Sender Chain Key continuously ratchets forward and the derived Sender Message Keys are never shared retrospectively, the new recipient or device can only decrypt messages sent after the SKDM is received, and all prior messages remain inaccessible to them.

Removal: Recipient or Device

When a recipient or device is removed, the Sender Key is reset on the next send. A fresh Chain Key and Signature Key pair are generated, packaged into a new SKDM, and distributed to all remaining recipients. The removed party receives no further key material and cannot decrypt any subsequent message.

Time-Based Expiry

Beyond membership changes, the Sender Key is also reset periodically based on how long it has been in use. Once the key reaches its maximum age, currently defaulting to 12 days and remotely configurable, it is automatically replaced. A fresh Chain Key and Signature Key pair are generated and distributed to all current

recipients via client fan-out. This ensures that even an active Broadcast List with no membership changes will periodically refresh its key material, limiting the impact of any long-term key exposure.

Security Properties

The following security properties are ensured by the Sender Key protocol employed for Broadcast Messages.

Forward Secrecy

Each Message Key is derived from the current Chain Key using a one-way HMAC function, after which the Chain Key advances, and the previous state is permanently discarded. Because the ratchet operates only in the forward direction, no prior message can be recovered even if the current key state is exposed; the ratchet cannot be reversed. Future messages are protected separately through Sender Key reset on membership changes and periodic expiry.

Post-Compromise Security

When a recipient or device is removed, the Sender Key is reset: the existing key is discarded, a fresh Chain Key and Signature Key pair are generated, and the new Sender Key is distributed exclusively to the remaining recipients through their individual pairwise sessions. This ensures that any key material previously held by the removed party, whether through legitimate access or compromise, cannot be used to read any message sent after that point. The same reset applies to time-based expiry, periodically restoring a clean key boundary even without a membership event.

Message Authenticity

Every message ciphertext is signed by the sender's Signature Key before transmission. Recipients verify this signature before attempting decryption. A valid signature cryptographically confirms that the message was produced by the legitimate sender's device and has not been modified in transit, protecting against forgery and impersonation by any party, including the delivery server.

Conclusion

Arattai's end-to-end encryption (E2EE) system, built on the Signal Protocol, ensures that only the intended users can access their conversations. By combining X3DH for secure session establishment and the Double Ratchet algorithm for dynamic key updates, Arattai achieves forward secrecy, post-compromise security, and asynchronous message delivery. The server functions solely as a relay and key distributor, with no access to private or session keys.

For any issues or security concerns, reach out to: incidents@arattai.in