Zhi White Paper

> Updated At 2025-05-24

Zhi: A Zero-Trust End-to-End Encrypted Messaging App.

Executive Summary

Zhi is a zero-trust end-to-end encrypted messaging app. Zhi addresses the challenge of enabling two parties to exchange messages via Zhi's servers without needing to trust those servers. It is designed primarily for users who rely on instant messaging to share information but cannot place trust in the IM service provider itself. Naturally, as an IM solution, Zhi ensures message reliability, ordering, and scalability—these standard features are not the focus here and will not be elaborated upon.

Background & Problem Statement

In today’s instant messaging market, even platforms that claim to offer end-to-end encryption often fail to enable true zero trust toward the service provider. Worse still, the concept of end-to-end encryption has been misused, misrepresented, and increasingly blurred, leading to widespread confusion among users.

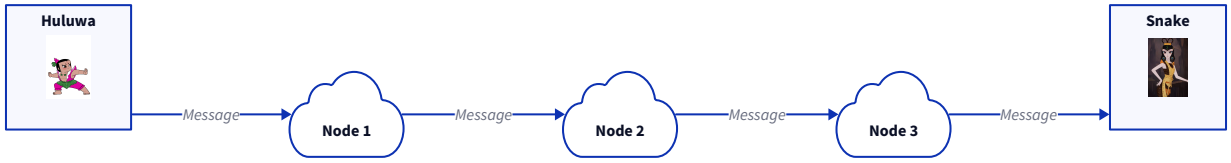
Here, we must ask a fundamental question:

Are WhatsApp, Signal, Telegram really end-to-end encrypted?

Let's break down this question step by step:

The first question: Huluwa and the Snake need to send messages through the network, what would the path of the message look like?

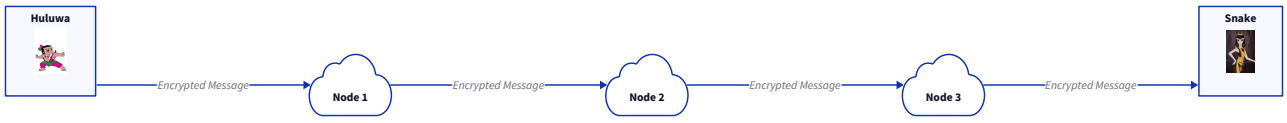
First of all, in today's internet based on TCP/IP, both Huluwa and the Snake must have their own IP to access the internet. But being on the network means that the messages from Huluwa or Snake may pass through many network nodes before reaching the other party. The intermediate network nodes could be internet service providers or IM service providers. See the illustration below:



It is obvious that the intermediary nodes would see the message, which is certainly not what Huluwa and Snake would expect. Therefore, they would consider encrypting the message before transmission. That leads us to the second question.

The second question: What is encryption?

Encryption is the process of using a **Secret** to encrypt a message into ciphertext, which can then be sent through any node without any issues, as long as the nodes do not have access to the secret and thus cannot decrypt the ciphertext to retrieve the message. The process is shown in the illustration below:



And now, the third question arises.

The third question: How can the Huluwa and Snake agree on this Secret?

That is, to encrypt messages before transmission, both parties need to agree on a Secret. One side will use this Secret to encrypt the message, then the ciphertext is sent over, and the other side, upon receiving the ciphertext, uses the Secret to decrypt and thus retrieve the message.

The issue is, how can both parties agree on this Secret?

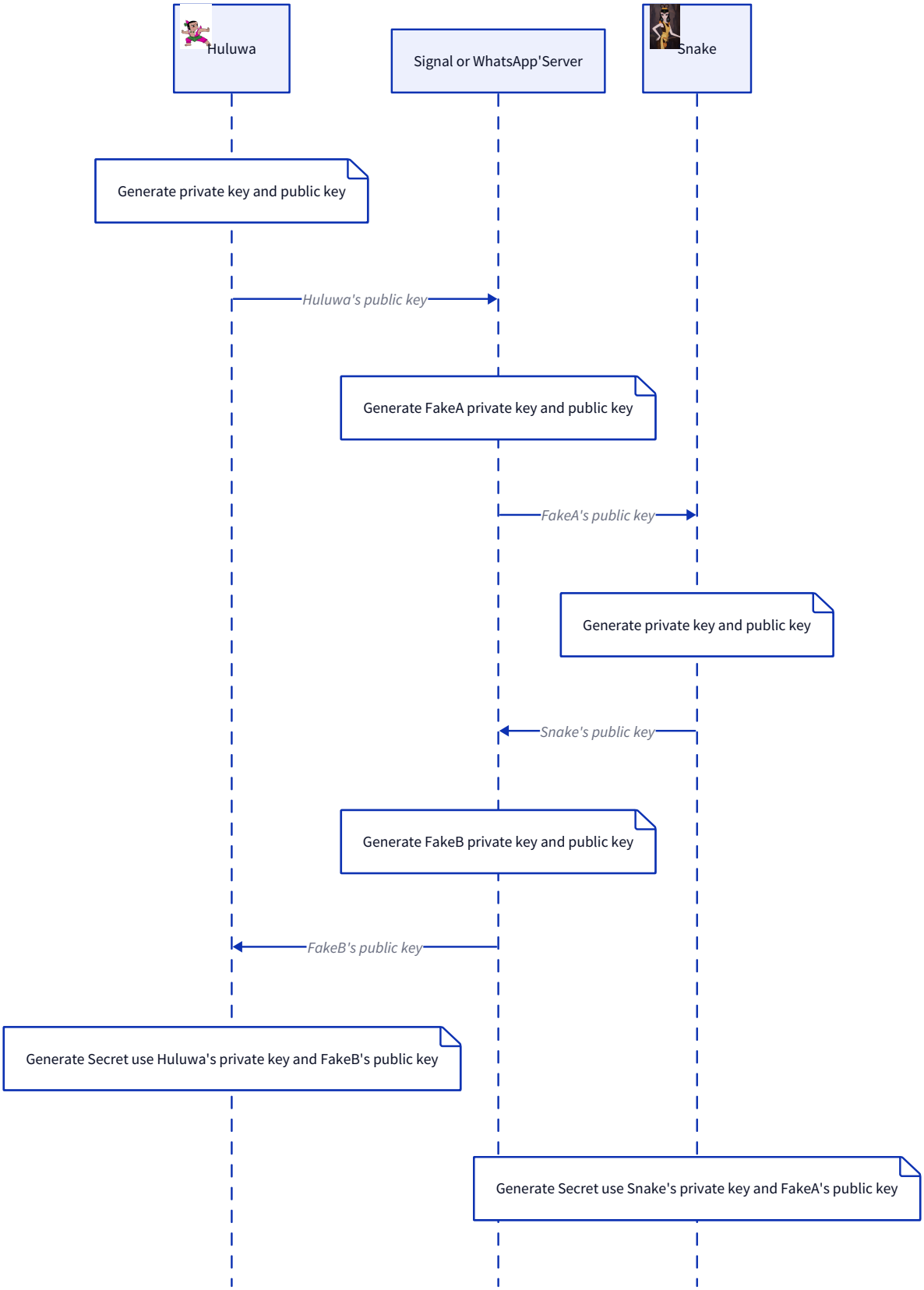
Currently, Signal and WhatsApp use the Diffie–Hellman key exchange algorithm to let Huluwa and Snake negotiate this Secret through Signal and WhatsApp's servers. Note that DH algorithm uses asymmetric keys, which is not the main point here. The general process is illustrated below:

1. Huluwa generate a pair of public and private keys.
2. Huluwa send their public key to Snake through the Signal and WhatsApp servers. Exposing the public key is not an issue.
3. Snake also generates a pair of public and private keys.
4. Snake sends their public key to Huluwa in the same manner. Exposing the public key is not an issue.

5. Huluwa use their own private key and Snake's public key to generate the Secret to be used.
6. Snake also uses their private key and the Huluwa' public key to produce the Secret to be used.
7. Finally, both parties have a Secret to use, or a Secret derived from it.

So, what's the problem with this? Long ago, I remember using mutt (a text-based email client for terminal), when people were into encryption, and if they wanted to securely send emails, they had to share each other's public keys in a secure way, such as offline, because any third-party method of sharing public or private keys was not reliable, including third-party public key servers. Back when Unix Hacker culture was prevalent, people even had to hold offline events to share public keys face to face.

Does Signal and WhatsApp have some new magic to solve this problem? After studying their papers and blogs, it appears they don’t. If this is unreliable, then the next steps in Signal and WhatsApp, such as X3DH, Double Ratchet Algorithm, forward secrecy, and other more complex concepts, all become unreliable as well. Because after going through the third-party role of Signal and WhatsApp servers, this third party can deceive both Huluwa and Snake. As shown in the illustration below:


Huluwa
Signal or WhatsApp'Server
Snake
Generate private key and public key
Huluwa's public key
Generate FakeA private key and public key
FakeA's public key
Generate private key and public key
Snake's public key
Generate FakeB private key and public key
FakeB's public key
Generate Secret use Huluwa's private key and FakeB's public key
Generate Secret use Snake's private key and FakeA's public key

That is to say, the Signal and WhatsApp servers can generate two pairs of public and private keys separately and interface with both Huluwa and Snake to generate a Secret with each. Huluwa may believe the Secret they generated belongs to Snake, and Snake may believe their Secret belongs to the Huluwa. Subsequently, when Huluwa and Snake send encrypted messages through Signal and WhatsApp servers, Signal and WhatsApp would be able to decrypt the transmitted messages.

So, how do Signal and WhatsApp solve this problem? The solutions are here:

- https://signal.org/blog/safety-number-updates/
- https://support.signal.org/hc/en-us/articles/6829998083994-Phone-Number-Privacy-and-Usernames-Deeper-Dive

This means you must confirm through a secondary channel outside of Signal and WhatsApp, such as offline, that the public key you're using is indeed the other party's public key. It ultimately comes back to this naive but essential question. As for Telegram, by default, it stores messages in plain text on the server, even if you activate encryption, it can't escape this essential issue.

The fourth question: Are WhatsApp, Signal, Telegram auditable?

Auditability means whether users can easily confirm that the software operates as claimed by its developers.

WhatsApp is not open source; Telegram's client is open source, but its server is not; Signal claims that both the client and server are open source.

- Then a question arises: Do we have any way to prove that the software downloaded from the Apple AppStore is 100% built from the open source code?
  - The answer is no.
- Another question is: Do we have the capacity to audit every line of code, especially when a software recursively depends on other codes?
  - The answer is also no, such as the recent malicious code in XZ Utils.

Apparently whether it is open source is not the main focus. So what should we pay attention to regarding auditability? We should focus on two points:

1. One is what data the software uploads to the provider's servers. The focus is on the uploaded data.
2. The other is that you should run the software in a sandbox environment, such as the AppStore, a browser, or higher versions of Android. This not only limits the software's permissions but also prevents other software from maliciously reading this software's data.

Regarding the first point, we can observe all the uploaded data of the software using network packet capturing tools (like mitmproxy or Wireshark).

- So, can we observe the network uploading data of WhatsApp, Signal?
  - After my testing, not ruling out the possibility of someone more skilled, without absolute certainty, my answer is that it’s very difficult.

Of course, being easy to observe also means being easy to crack, so from a certain perspective, it's understandable.

So,

We need a zero trust model. This means:

1. **Message encryption keys must never be transmitted through the IM server.**
2. **The IM client must encrypt all messages with the key before sending them to the IM server.**
3. **All data sent from the IM client to the IM server must be easily auditable.**

Product Overview

Zhi is a zero-trust end-to-end encrypted messaging app.

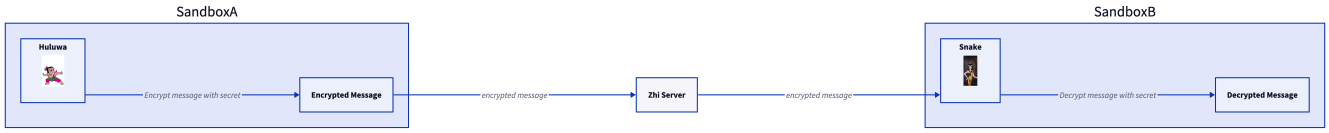
- Encryption keys are shared directly between users, either face-to-face or through trusted out-of-band channels.
- All messages are encrypted with the key before being transmitted to the Zhi server.
- All files are also encrypted with the key before being transmitted to the Zhi server.
- Each chat session is isolated and establishes a new identity for the user.
- Random UUIDs are used as metadata to enhance privacy.
- All data sent from the Zhi client to the Zhi server is designed to be easily auditable.
- The Zhi client runs within a sandbox environment trusted by the user.
- Zhi supports the use of BTC (Bitcoin) to create anonymous accounts.
- Audio and video communication in Zhi also follows the zero trust model and ensures that your IP address is never exposed to the other participant.

Technical Architecture

This section focuses solely on the zero trust model. For a more detailed overview of Zhi’s full system architecture, please refer to the official Zhi website.

The diagram below illustrates Huluwa and Snake performing a face-to-face key exchange.

The diagram below shows Huluwa sending a message to Snake via the Zhi server.



Auditable

The data sent from the Zhi client to the Zhi server includes randomized UUIDs and fully encrypted messages, ensuring privacy and preventing correlation or tracking.

All images, videos, and files are also encrypted with the key before being transmitted to the Zhi server.

We use standard WebRTC technology to implement Meeting. We know that WebRTC is end-to-end encrypted, STUN and TURN will not break WebRTC's end-to-end encryption. And we enforce the use of TURN, so when you have a Meeting with other members, the other members are not able to know your IP. Because the key negotiation for WebRTC is achieved through signaling interaction. Therefore, it's essential to ensure the security of the signaling being transmitted. Don't worry, all your signaling, just like the messages above, is also encrypted with your Chat Key before transmission.

Chats are isolated from one another, meaning that even if Huluwa and Snake are both in Chat A and also in Chat B, they cannot know that they are in both chats at the same time.

What about Push Notification?

As we know, as an instant messaging app, we need to implement push notifications. On the Apple platform: We must choose Apple APN (Apple Push Notification service). On the Android platform: We almost have no choice but to use Google FCM (Firebase Cloud Messaging). On the Web platform: Currently, we are only developing and doing compatibility testing on Chrome. To implement push notifications, we must use Google GCM (Google Cloud Messaging). Accessing: chrome://gcm-internals/. Because Zhi server cannot decrypt messages, when there is a new message, we can only push a bland "New Message" string to the push service, which then sends it to the user's device. Therefore, neither Apple nor Google can decrypt your messages.

Using Apple's IAP, will my information be exposed?

No. We only pass a transaction ID upstream.

Zhi adopts a transparent WebSocket-based transport protocol with JSON-encoded payloads. All client-side network interactions are designed to be easily auditable, and full upstream request documentation is available for verification and analysis.

https://www.txthinking.com/talks/articles/zhi-upstream-api-documentation-en.article

Enjoy Zhi

https://www.txthinking.com/zhi.html