The Labyrinth Encrypted Message Storage Protocol

December 6, 2023
Version 1

Meta

Abstract

End-to-end encrypted messaging creates significant new challenges regarding data storage and accessing message history between devices. Labyrinth is a novel storage system, currently being developed and rolled out in Messenger, which aims to serve this purpose while maintaining a high bar for message content privacy. It is designed to protect messages against non-members (devices and entities which are not enrolled in a user's Labyrinth mailbox) , including preventing new messages from being decryptable on revoked devices which may have previously had access to earlier messages, while achieving low operational overheads and high reliability.

This document describes the Labyrinth protocol as designed, and as it is intended to operate. Details of Meta's implementation may differ in places and will likely evolve over time. This white paper should not be read as making any assurances or commitments to users on Meta's products or services.

1. AES-GCM-Extended

AES-GCM-Extended is the approach that Labyrinth follows to avoid nonce reuse concerns. It takes inspiration from XChaCha20-Poly1305 in its approach, but uses AES-GCM-256 as its underlying cipher, as this is already present and used elsewhere in Messenger's codebase.

AES-GCM-Extended takes a 32-byte key and a 28-byte nonce. From these it generates a subkey and subnonce, as follows:

```
subkey := hchacha20(nonce[0:16], key)
subnonce := nonce[16:28]
```

These are then used directly within AES-GCM-256 to encrypt/decrypt the data. The 28 byte nonce is prepended to the ciphertext.

2. Padding

When Labyrinth encrypts messages using AES-GCM-Extended, padding is also applied to protect ciphertext lengths. Padding involves a tradeoff between privacy and storage overhead: longer padding hides message lengths better, but uses more storage space. We use the PADMÉ scheme[3], which provides a good balance.

For messages with fewer than 10 characters, the plaintext is simply padded to 10 characters. Otherwise, the PADMÉ scheme is applied. This scheme bounds leakage to no more than `O(log log N)` bits for messages of length `N`, and uses at most around 12% overhead.

Each message plaintext is prefixed with a 4 byte unsigned big-endian integer indicating the length of the unpadded plaintext.

[3] https://nikirill.com/files/purbs.pdf