Kursal: A peer to peer encrypted messenger

Erik LP.
Kodeur_Kubik
kubik@openvoxel.studio

Armand H.
Arlo
arlo@openvoxel.studio

Abstract

Kursal proposes an end-to-end encrypted, peer-to-peer protocol for exchanging messages and files over a decentralized network. It uses Signal's encryption methods (Double Ratchet, PQXDH) combined with libp2p's protocols to achieve a fully decentralized setup. The network relies on volunteering nodes: the more nodes that are connected, the more robust the network gets. Any node can offer itself as a public relay to help others connect across NATs or behind firewalls. Kursal's objective is to combine security, decentralization and usability into one unified application.

Version: 1.0
Date: December 24, 2025
Website: https://kursal.openvoxel.studio/

1. Introduction

Messaging platforms have become the most widely adopted systems for online communication and file transfers. However, commonly used messengers either do not encrypt messages on their servers or implement end-to-end encryption imperfectly. Others compromise user privacy by collecting private user data and using closed-source clients. Even transparent systems are vulnerable. Governments could still access centralized hosting servers, block or filter content, censor users, or spy on them. This is where Kursal steps in as a potential solution: all messages, files, and calls are end-to-end encrypted and transit over a decentralized network hosted by volunteers.

1.1. Motivation

Growing centralization in digital communication threatens user privacy and autonomy. Proposals like the European Union's Chat Control initiative to scan user messages for harmful content highlight the risk of surveillance becoming normalized under the guise of safety. Kursal aims to demonstrate a different path: a fully decentralized, peer-to-peer protocol for private communication based on modern cryptography. We believe privacy and security can coexist through open, verifiable, and community-driven technology. We do not endorse crimes but believe in non privacy-intrusive ways of countering them. Our goal is to demonstrate that decentralized peer-to-peer encrypted messaging can achieve the same level of usability and reliability as centralized platforms.

2. First Contact

There are three ways for two peers "Alice" and "Bob" to initiate a communication over the Kursal network. Each method has its qualities and disadvantages:

- **One-Time Password**: Easy to share. Takes time to publish on the network, is single-use and requires computation.
- **Long-Term Code**: Multiple usage code that requires no computation. File size of 7 KB that must be transferred via an external channel.
- **Nearby Share**: Easy to share. Devices must be on the same WiFi network (which must support mDNS) to initiate the connection.

2.1. With a One-Time Password

Alice generates a one-time password (referred to as OTP) composed of 6 digit-word pairs. The interface can also display this code as a QR-code that can be scanned by the other user. The word dataset comes from https://www.eff.org/dice and we filtered it which ends up with a total of around 7500 words that are as short and as simple to write as possible. Alice computes the argon2id [1] hash of this OTP (referred to as `OTP_H`). For benchmarks and information about this hashing configuration, refer to Section 6.4. An initial PQXDH [2] payload is encrypted with

argon2 and AES-256 (with an empty salt). It is then published to the distributed hash table (DHT) [3] under the hash `OTP_H` and argon2 encrypted value `(PQXDH_payload, Peer ID, pubkey, relays)` with the password OTP. The DHT entry expires and is automatically removed after 10 minutes. The public sign key (`pubkey`) is a public Dilithium-5 [4] key that will later prove the integrity of messages. The `relays` entry contains addresses on which Alice can be contacted (direct and relays). Bob may now fetch the corresponding DHT entry by first hashing the OTP he received from an external source. He can then complete the PQXDH protocol and initiate a Double Ratchet [5] extended by the ML-KEM Braid Protocol [6]. For Alice to also complete the PQXDH protocol, Bob must send the final PQXDH payload, his public signing key, his public listening addresses (like Alice's `relays`) and a Double Ratchet initial message through a relay.

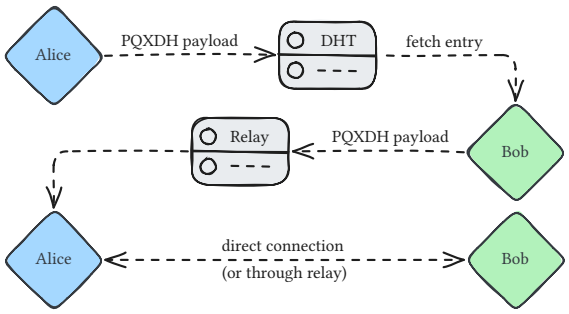

Figure 1: Direct contact with a OTP

2.2. With a Long-Term Code

Exchanging individual one-time passwords can be time-consuming. To simplify managing multiple friends long-term, Alice creates an object containing her peer ID, an initial PQXDH payload [2], an ephemeral public key, a public Dilithium-5 [4] signing key as well as public addresses to reach Alice (direct IP or via relays). The PQXDH payload does not contain a one-time pre-key as this payload can be used multiple times by different users. The ephemeral key is kept in cache by Alice and deleted once the Long-Term Code (referred to as LTC) is renewed or invalidated. This object can then be encoded into a ".kursal" file (which weighs approximately 6 KB). After receiving the file, Bob decodes this object and completes the PQXDH protocol by sending the PQXDH payload, his public signing key and his public listening addresses to Alice. They can initiate a Double Ratchet [5] extended by the ML-KEM Braid Protocol [6] as described in Section 2.1.

2.3. Nearby Share

The same process as the Long-Term Code (Section 2.2) can be repeated but instead of sending a file over internet as described previously, it can directly be shared to nearby devices via multicast DNS (mDNS). A local record of all mDNS reachable devices can be kept in memory and used to contact peers on the same network. This process could be extended with Bluetooth but is not implemented in our first version of Kursal.

2.4. Security Code

Each security code is unique to each contact and should be verified as soon as the communication is established. It should match on both sides and if not, it very likely means a man-in-the-middle attack is happening. Security codes should **not** be verified by sending them over the chat because in the case of a man-in-the-middle attack, the attacker could impersonate this code as well.

It is computed by key deriving the identity keys and signing keys of both parties in a deterministic way in order to get the same result on both sides. Both public identity keys are sorted and concatenated with the two sorted Dilithium-5 public signing keys. Those four concatenated keys are then hashed with the SHA-256 algorithm. The output is then formatted as 8 groups of 4 digits from the SHA-256 and displayed on the user interface.

3. Continuous Communication

Kursal is based on rust libp2p's swarm architecture. It supports both TCP and QUIC protocols and automatically finds the optimal networking solution. Each connection initially starts from a relay and will, if possible, be upgraded using DCUTR (Direct Connection Upgrade Through Relay). This upgrade can be a direct peer-to-peer connection or both peers can attempt a NAT traversal using hole punching. If both of those fail, the connection is maintained through the relay. Each node is identified by its peer ID, which rotates regularly to prevent tracking.

If a connection drops (e.g. network switch), the system automatically reconnects peers via available relays. For instance, switching from Cellular to WiFi will momentarily stop the communication that will be automatically resumed. During the initial communication and on changes, both peers know addresses their contacts are listening on.

Each message receives a reply containing the Dilithium-5 [4] signature of that same encrypted message to prove authenticity. This signature should be verified, and the message should not be marked as delivered until Alice's signed reply is received.

Figure 2: Signed reply of a received message

4. Data Streams

In order to transfer a large amount or a continuous stream of data like file transfers and calls, it cannot encrypt each packet with Double Ratchet as this method would be too heavy in computing. Therefore, each time a data stream is about to happen (calls, file transfers, ...), both agree on a random and temporary key to encrypt all following packets using the AES-GCM [7] protocol. This allows lightweight encryption for one-time transfers. Each peer generates a random 32-byte key that is sent to the other via Double Ratchet. Both peers then derive a new key via XOR of their two keys, ensuring a mix that cannot be guessed by compromising one of the original keys.

5. Implementation

We propose a Kursal implementation written in Rust available at https://github.com/KursalChat/Kursal under the GNU AGPLv3 license, available on Linux, Mac and Windows. Our implementation uses Signal's libsignal and libp2p's rust-libp2p libraries.

We plan on improving this initial version over time by adding offline messaging support, multi-device support, groups and broadcasting channels. These features are listed on our TODO list and will be implemented over time.

6. Risk Mitigation

6.1. Operating System

If the operating system itself is compromised, Kursal cannot guarantee the integrity of messages. All encryption keys, which are stored in the keychain of the computer could be accessed by malware or by the operating system itself. It is up to the user to choose a trusted operating system and keeping it secure.

6.2. Malicious Relays

Relays, even malicious, will never be able to read your messages as they are encrypted. However, they can analyze your messaging patterns if unprotected. Therefore, peers will try to use "multi-hop" relays, meaning your data will go through multiple relays before reaching the destination like the TOR network. Those relays could track your peer ID and who you're talking to. This is why peer IDs rotate regularly to prevent relay operators from tracking your communication patterns. Additionally, relays have access to your IP address if you are not using a VPN.

6.3. DHT Interception

Intercepting the DHT is defined by accessing an encrypted initial PQXDH payload by finding the OTP. In a hypothetical scenario, a hacker could pre-compute all pairs of OTP and their hash and decrypt the payload. The initial payload does not contain any sensitive information as it is only made up of public keys. The hacker could attempt a man-in-the-middle attack but it could be detected by a different security code (see Section 2.4).

6.4. DHT Brute Forcing

To evaluate possible risks of pre-calculating every possible OTP and its corresponding hash, we can estimate how long it would take to compute. Given a computer hashing speed of $CS = 1000$ hashes per second and per device, a time period of 1000 years, which approximates to $T \approx 3 * 10^{10}$ seconds. The number of possible combinations of the OTP is $P = (10 * 7500)^6 = 1.8 *$

10^{29}. We can approximate the number of devices D needed with this hashing speed to calculate every possibility:

$$D = \frac{P}{\mathrm{CS} * T} \approx 6 * 10^{15} \text{ devices}$$

This means that in order to intercept 0.1% of the DHT entries, you would have to calculate at 1000 hashes per second for 1000 years with $6 * 10^{12}$ devices. And for one interception in a million, it drops to 6 billion ($6 * 10^{9}$) devices for the next 1000 years at 1000 hashes per second. For more details about what DHT interceptions lead to, refer to Section 6.3.

The Argon2id configuration used is set to last a significant time on modern computers. We are therefore well above the theoretical 1000 hashes per second, making brute-forcing impractical and future-proof. Here are the results of the hashing benchmark (Kursal uses those exact settings) with: memory = 16 MiB, iterations = 32, parallelism = 4.

CPU	Time per iteration	Iterations per second
Intel Core i5 14600kf	583 ms 29.2 ms (threaded)	34.15
iMac M4	239 ms 30.0 ms (threaded)	33.32
Intel Core i5-10310U	629ms 78.9ms (threaded)	12.69
Intel Core i7-4770	963ms 241 ms (threaded)	4.15

Table 1: Hashing Benchmark: 1000 iterations

The taken CS = 1000 hashes per second is an extremely high speed that is not reached by modern computers. It is therefore theoretically possible to brute force the DHT, but very unlikely. If an attack of some sort happens, the Argon2id salt could be changed from null to a random one and the attackers would have to recalculate all the possibilities.

Bibliography

[1] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: the memory-hard function for password hashing and other applications," technical report, Mar. 2017. [Online]. Available: https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf

[2] E. Kret and R. Schmidt, "The PQXDH Key Agreement Protocol," technical report, May 2023. [Online]. Available: https://signal.org/docs/specifications/pqxdh/

[3] Raulk, J. Hiesey, and M. Inden, "libp2p Kademlia DHT Specification," technical report, Dec. 2022. [Online]. Available: https://github.com/libp2p/specs/blob/master/kad-dht/README.md

[4] L. Ducas *et al.*, "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme," technical report, Feb. 2021. [Online]. Available: https://eprint.iacr.org/2017/633.pdf

[5] T. Perrin, M. Marlinspike, and R. Schmidt, "The Double Ratchet Algorithm," technical report, Sept. 2025. [Online]. Available: https://signal.org/docs/specifications/doubleratchet/

[6] R. Schmidt, "The ML-KEM Braid Protocol," technical report, Feb. 2025. [Online]. Available: https://signal.org/docs/specifications/mlkembraid/

[7] D. McGrew and K. Igoe, "AES-GCM Authenticated Encryption in the Secure Real-time Transport Protocol (SRTP)," technical report NIST Special Publication 800-38D, Oct. 2008. [Online]. Available: https://datatracker.ietf.org/doc/rfc7714/