Versioned Storage Service (VSS)
Useful for synchronized backup/access of bitcoin wallet and LN state data

Idea

A LN-based bitcoin wallet has a lot of state that needs to be securely stored. This is necessary for a couple of reasons:

1. If multiple devices are accessing the wallet, they need a way to access and share LN state without corrupting the state.
2. Recovery. If a user loses their phone or otherwise loses access to their app/app's data, they'll need to restore the state in order to continue to have secure access to their money.

Since nearly every non-custodial LN wallet will need this, every wallet dev will need a solution. There is opportunity here to create a reusable component that all wallets can use and reduce the barriers to building a great LN wallet.

This solution should also be able to work for any LN implementation that supports K-V store. This currently includes LDK, VLS, and Greenlight+VLS. It might accommodate LND in the future.

Tentative Requirements

1. An encrypted blob of data needs to be stored in the cloud
2. Multiple devices should be able to access this (read/write) and there needs to be a mutual exclusion synchronization method to prevent multiple devices from updating state at the same time
3. Rate limiting/DoS protection (so bad actor with a wallet cannot upload massive encrypted files)
4. Support some form of encryption to encrypt data to be uploaded to cloud
5. Support any LN state machine that uses K-V store (LDK, VLS, etc.)
6. Have API to allow devs to use any cloud storage solution
7. Support multiple cloud providers out of box (AWS, Google Cloud, Azure?)
8. Should be able to host this data standalone or part of another service like a watchtower or LSP
9. Support any data that needs to be backed up including channel data (for LDK, ChannelMonitors and ChannelManager), routing/velocity controls (for VLS), app/user config data, derivation paths, wallet birthday), payment history, bolt12 subscription data, etc.)
10. Client-side code should be as compact as possible as it may need to run in low-resource HSM environments OR client-side could should be able to be run server-side in which VLS/embedded environment can communicate with
11. Open question: Is the scope of this only individual users or also enterprise?
12. Prevent rollback, because this would break the Lightning trust model. This would require the ability to use multiple providers of this service and ensure that no provider is

supplying old data. Need a protocol to commit to the multiple providers in an atomic way.

Design Considerations

1. How does an app authenticate to the server that is has authorization to read or write/update the encrypted blob?
2. What is the mutual exclusion synchronization method used to ensure 2 apps aren't updating LN state simultaneously? Solutions that have been tossed around include optimistic locks, pessimistic locks, etc. Also let's confirm that iCloud/Google Drive do **not** support such mechanisms so we can rule them out as options for this.
3. How many encrypted blobs are there? One idea is to have an encrypted blob per channel in order to increase concurrency between different apps accessing the same wallet. This is probably overkill for an individual user use case (who might only have 1-3 channels) but could be material for enterprise use case.
    a. Also discussed was the idea to store some data in the users iCloud/Google Drive…why add this complexity? Yes it is under the control of the user is already trusting the cloud service storing the LN state or else they can lose their money. It seems reasonable to also trust it that it will retain things like user config and payment history?
4. Which encryption method to support? AES or ChaCha20 or something else?
5. How to handle rate limiting/DoS protection, and whether to do anything in v1
6. How many cloud providers (AWS, Google Cloud, Azure, etc.) do we support out of the box in v1? Long-term? We need to better understand the eng effort to build and maintain each of these. If non-negligible then probably just start with AWS.
7. If the scope includes enterprise use cases, how much additional engineering work is required to handle cases with orders of magnitude more data stored (thousands of channels and million of payments in history)?
8. Is interfacing with cloud providers (like AWS) done client-side or server-side? If client-side, what are the security implications of users having access to AWS? Why *wouldn't* we do this server side?
9. If we do client-side calling into AWS, etc. then which language do we write the code in, since apparently AWS's Rust support is only beta and Google Cloud doesn't support Rust?
10. Which repo/crate does this code live? If it is intended to support not just LDK but also VLS, etc. should it be in a non-LDK independent GitHub org/crate?
11. A security concern is a malicious server serves an old state (encrypted blob). The current recommended solution is the app needs to store the data on 2 independent servers and use whichever serves up the most recent state (in general they should match).
12. Distributed transactions: for VLS+node, or multisig, etc., what happens if client stores the blob, but the Lightning node crashes before persisting its state? The node might not replay the same thing when it comes back up

References

1. Blue Wallet implemented an encrypted blob LN state cloud solution. It doesn't support multi-device synchronized access. https://github.com/BlueWallet/bytes-store
2. Photon is related and likely complements this solution. https://photonsdk.org/