Metaphi: Non-custodial wallet with a custodial experience

May 28, 2022

www.metaphi.xyz

Abstract

We present an architecture for a chain-agnostic and non-custodial crypto wallet. Our solution does not require users to memorize long key recovery phrases or depend on third party HSM providers. Account recovery is facilitated via a semi-trusted third party. Fund and token access is mediated between the dApp and the user directly. Our design relies on thresholding based secret sharing and is similar to ‘multisig’ approaches.

Contents

Introduction and existing work

Crypto wallets are available in two categories - custodial and non-custodial. In the former, the private keys required for signing transactions are held by a trusted third party. In the latter, the private keys are stored locally on the user's device, generally encrypted at rest, and recoverable from a 16 word phrase which maps to the 32 byte private key. These two approaches affect three key user facing aspects of wallets; compliance requirements, security and UX.

Custodial

Custodial wallets provide the simplest sign-in experience for non-crypto native users via familiar Web2 based flows. They have proved to be the easiest way to on-ramp users into the crypto space and also allow for simple developer integrations. There exist several of these in the market, such as Coinbase, Venly and Robinhood. However, custodians have to comply with stringent KYC and AML laws which requires a multi-step initial sign up flow. Also, hosted wallets provide a singular target for attackers looking to exploit security flaws. Custodian wallets in the past have a history of having funds stolen [1][2][5].

Non-Custodial

Non-custodial wallets are used by sophisticated, crypto native users. These are generally available as a browser extension, the most popular being Metamask. While

[1] The $800 million Bitcoin wallet no one wants to touch - Decrypt
[2] How DOJ Tracked Down the Bitcoin Stolen in Bitfinex Hack | Time

most DApps have integrations with Metamask, we have found that non-crypto native users find Metamask intimidating, resulting in significant user drop off.

Some wallets delegate signing to a third party cloud provider's KMS solution or a trusted execution environment (TEE). Keys are held in a HSM, which themselves are secured in the provider's data center. While delegation of signing to a cloud provider absolves the wallet provider from holding the private keys, they still need to manage their interactions with said provider, leaving open the malicious insider attack vector. All cloud providers offering KMS solutions today use the same vendor (Marvell) to source their HSMs[3]. This makes such wallets highly susceptible to supply chain attacks.

The idea of using threshold based secret sharing to manage private keys is not novel. Some solutions exist where one of the shares is stored on a user device but the other two shares are distributed in a decentralized network amongst a set of validator nodes operated by well known crypto native entities[4] and downloaded to a secondary device, which could be a phone or a desktop.

In contrast, our protocol is simpler and does not require a set of trusted node operators. Trustless operations are necessary to mitigate insider risk. Moreover, instead of increasing user burden by requiring them to have a secondary device or remembering answers to secret questions, we give them the option of storing a part of the secret with a semi-trusted third party.

---

[3] (38) Risks in Cryptography in the Cloud | LinkedIn
[4] List of Node Operators | Documentation

Multisig wallets

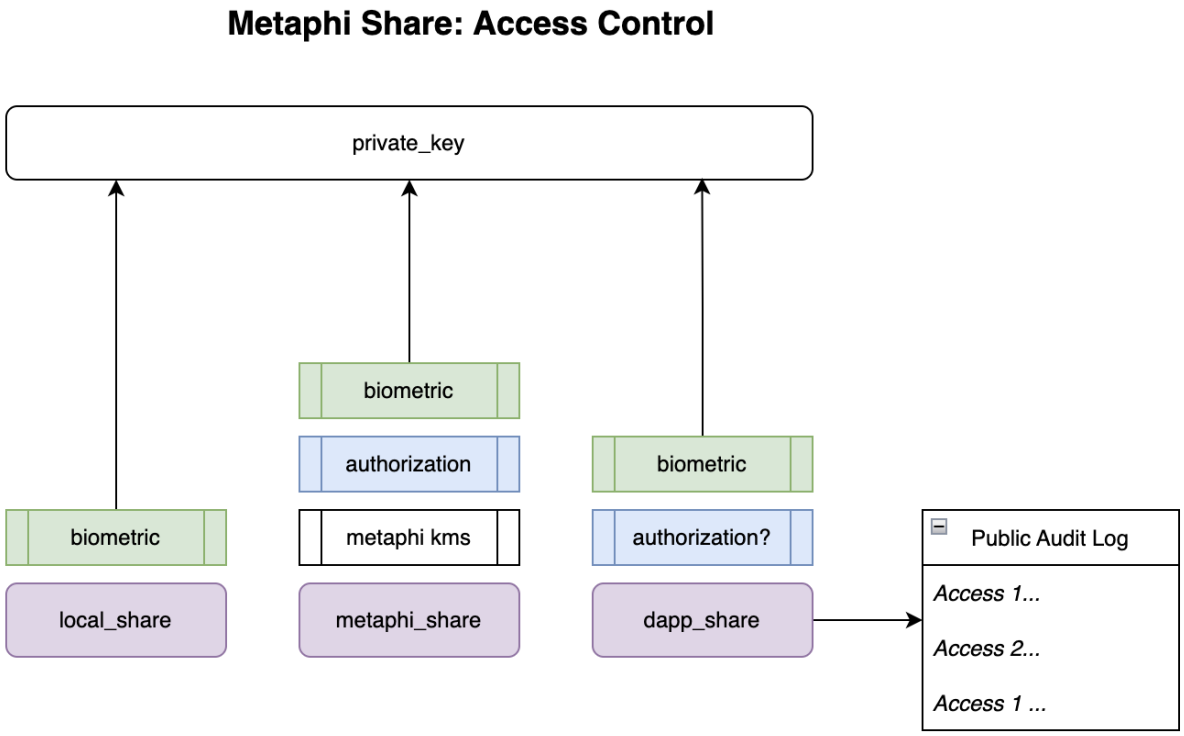
Multisig is not a native concept in Ethereum and multisig wallets are implemented as smart contracts[5]. Several quality implementations exist and are a good use case for securing large sums of cryptocurrencies for users desiring strong security. However, multisig wallets are inappropriate for many use cases, especially DApps that require frequent transactions such as gaming.

Multisig wallets use ECDSA based threshold signing, where m of n signatures are required for a transaction to be processed. Even though significant work has been done improving these signing algorithms, they are still quite slow. Moreover, the runtime of these algorithms increases linearly with the number of participants.

One of the fastest algorithms[6] is benchmarked at O(100ms) with just two participants on an implementation that does not take into account network round trip times and latency. For applications that require frequent signing, in real world conditions, this will result in degraded user experience. Liveness is another issue, if one of the signing keys is on another device, both will need to be available for signing.

[5] Overview - Gnosis Safe (gnosis-safe.io)
[6] Gennaro, R. and Goldfeder, S. Fast Multiparty Threshold ECDSA with Fast Trustless Setup, Crypto 2019.

Metaphi Non-Custodial Wallet Solution



Link: Flowchart Maker & Online Diagram Software

Design

When a user creates a Metaphi wallet, we generate the public/private key pair locally on their device. Verifiable secret sharing (VSS)[7] is used to create secrets from that private key. These secrets are then encrypted using a user-provided pin and

[7] Feldman, Paul (1987). "A practical scheme for non-interactive verifiable secret sharing". 28th Annual Symposium on Foundations of Computer Science (SFCS 1987): 427–438.

distributed to several entities described in the section below. In subsequent sections, we describe how these secrets are managed and secured against adversaries.

Metaphi pegs the wallet address and key retrieval methodology to a single piece of user information, which is customizable based on the preference of the dApp. The available and planned methods are:

- User Email
- User Phone Number (planned)
- ENS aka Ethereum Name Service (planned)

When the user wants to retrieve their private keys for signing transactions, they will have to verify their information, which will allow them to retrieve their secrets. Finally, they will use their pin to decrypt the secrets allowing a complete reconstruction of the private key.

Distribution of Key Holders

There are three holders of the generated shares. The three stakeholders represent each involved party in an authorized transaction.

User Device

The user device is used for two key functions

- Stores part of the secret share
- Encrypt all shares, with a biometric-protected symmetric key. This is described in detail below.

The reconstruction of the private key always happens on the user-device itself, thereby establishing the non-custodial nature of the wallet. To ensure this, all keys are encrypted with the users' symmetric key, protected by biometric credentials at a top-level.

DApp operated Smart Contract

This is a smart contract deployed on a public blockchain, that stores secret shares for the DApp's users. The share present on the chain is encrypted twice, once using the user's biometric credential-based encryption key, and then again by the DApp's encryption key.

The purpose of storing it on-chain is to provide high availability for the secret managed by the DApp. Therefore, even if Metaphi is unavailable, the user can continue interacting with the DApp.

A strawman implementation of the smart contract storing the secret is as follows:

```solidity
// SPDX-License-Identifier: LGPL-3.0-or-later
contract MKMS {
  address internal deployer;
  mapping (address => Secret) internal addressToSecret;
```

```
  mapping (address => uint256) internal addressToLastAccessTime;


  constructor()
  {
    deployer = msg.sender;
  }


  function addAddress(address requestor, Secret secret)
    external
  {
    require(deployer == msg.sender);
    addressToSecret[requestor] = secret;
  }


  function secret(address requestor)
    external
    payable
    returns (Secret memory _secret)
  {
    addressToLastAccessTime[requestor] = block.timestamp;
    return addressToSecret[requestor];
  }
}
```

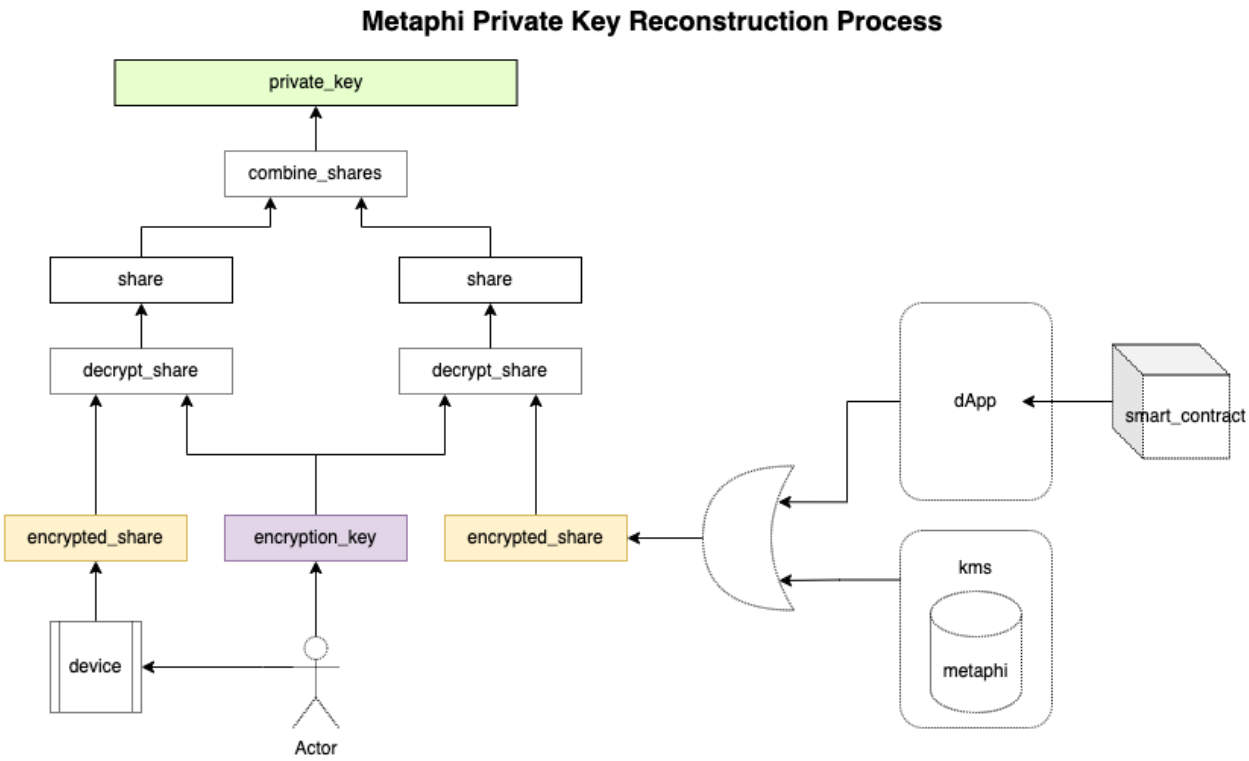
```

The contract solves two purposes:

- Allows for public auditability and access trails for the dApp’s secret share piece since all interactions with the smart contract is necessarily logged with a transaction hash available on a block scanner.
- Allows dApp to host the users’ secret share without accountability in a crypto native and decentralized manner.

When the user first interacts with the DApp and successfully authenticates themself, the DApp will fetch the secret from the contract, decrypt it using its own encryption key and pass it on to the user. That secret is then further decrypted using the users symmetric key. Combined with the share on the user’s own device, their private key can then be reconstructed locally. Note that at no point is Metaphi's involvement required for normal operations.

Access to this on-chain contract can be done via a dockerized service, which could have automatic key management and rotation built in, simplifying operational overhead for the DApp.

Metaphi (semi trusted third-party)

Metaphi Database stores the mapping of the secret shares based on
```

- chainId
- dApp
- UserId (described above)

The secret share is encrypted twice

- Biometric-credential protected key from the user
- Metaphi's Cloud KMS encryption

Metaphi DB only acts as an account recovery mechanism, and in the ideal scenario, this share is never used to generate private keys.



Link: Flowchart Maker & Online Diagram Software

Biometrics secured encryption key

All shares are encrypted by a user-level, non device-specific key. This is intended to be a biometric-based key or simple pin protected by device-based biometrics.

This is a symmetric key which means the same key is used to encrypt and decrypt the shares. Since this key never leaves the users' device, it can be ensured that the reconstructing of the encrypted private key shards can only be possible on the users' device. As the key is non-device specific, it may be reconstructed in the event the original user device is stolen.

Browser

Creation of the symmetric key for web apps can be done easily using a hash of a user provided pin and their email address. This can be further secured using a 2FA device like a security key.

Mobile

For mobile devices, this approach works in a similar manner, with the key being secured using techniques like FaceID or fingerprinting.

Proof of non-custody

Which combination of parties can regenerate the secret key?

Party 1	**Party 2**	
User Device	Metaphi	Yes
User Device	dApp	Yes
Metaphi	dApp	No. User biometric-credential is missing

The above truth table demonstrates that the Metaphi wallet is completely non-custodial and ensures that the user is always the only owner of the reconstructed private key. Despite the key shards being distributed across multiple stakeholders, without the user's biometric-credential, the shards would be useless.

Features

Chain Agnostic

Our approach can work across arbitrary L1s and L2s, as long as they support the smart contract functionality our protocol requires.

Account Recovery

Only 2 out of 3 shares are required to regenerate the private key for the user. Incase of loss of the users' key, Metaphi provides access to the backup share.

The backup share is protected in three ways

- Encrypted by users' biometric credential generated key
- Encrypted by Metaphi's Cloud KMS
- Whitelisting of domains which can request for Metaphi's share

Note that while this scheme can be easily extended to more than 3 shares, we believe a simpler solution is warranted in the shorter term.

Account Protection

Accounts are protected by the fact that there are three key-holders, and multiple levels of encryption at each level. The table below outlines the lines of defense for each:

User Device	Native Device Protection (Device lock, screen lock, app lock)
	User Biometric based symmetric encryption
	Limited persistence for generated private key via scoping it's lifetime to an active session.
dApp Contract	User Biometric based symmetric encryption
	Auditable, access is publicly trackable
	Contract level access to information to contract deployer (dApp); Since the information is public, data could still be downloaded

	and read, however, this condition adds an additional deterrent
	Encryption layer (optional); Stored shares can be encrypted via the dApp itself and decrypted in a backend service, before being passed to the user
	Whitelisting of domains (optional); If the dApp chooses to host a backend service, the access to contract credentials can be controlled by whitelisting domains from which request to access this contract can be made
Metaphi Database	User Biometric based symmetric encryption
	dApp domain whitelisting; Metaphi only allows access requests from pre-configured whitelisted domains
	Cloud-based KMS to encrypt hosted shares

Native Experiences

By providing an open-source SDK to enable this mechanism, we allow dApps to embed a native wallet experience in their apps. The third-party, described in this paper, as the Metaphi database, could be any third party, including the dApp itself. However, in this case, the dApp would carry greater liability owning 2 out of 3 shares. However, given the mechanism uses a top-level encryption sourced from the users' biometric,

even if the dApp owns 2 out of 3 shares, they will not be able to regenerate the secret key by themselves without the user's authorization.

Account Syncing Between Devices

For example, the user uses a dApp both on the browser as well as the phone. If the user first logged-in to the app using the browser, the secret fragment will continue to live on the browser. To allow the user to authenticate via their phone, the shards from the dApp and Metaphi will be used to reconstruct the private key on the user device.

In a future development, we could consider more than 3 shards, to allow for the user to use Metaphi only as a recovery mechanism and provide the ability to host shards on multiple user devices.

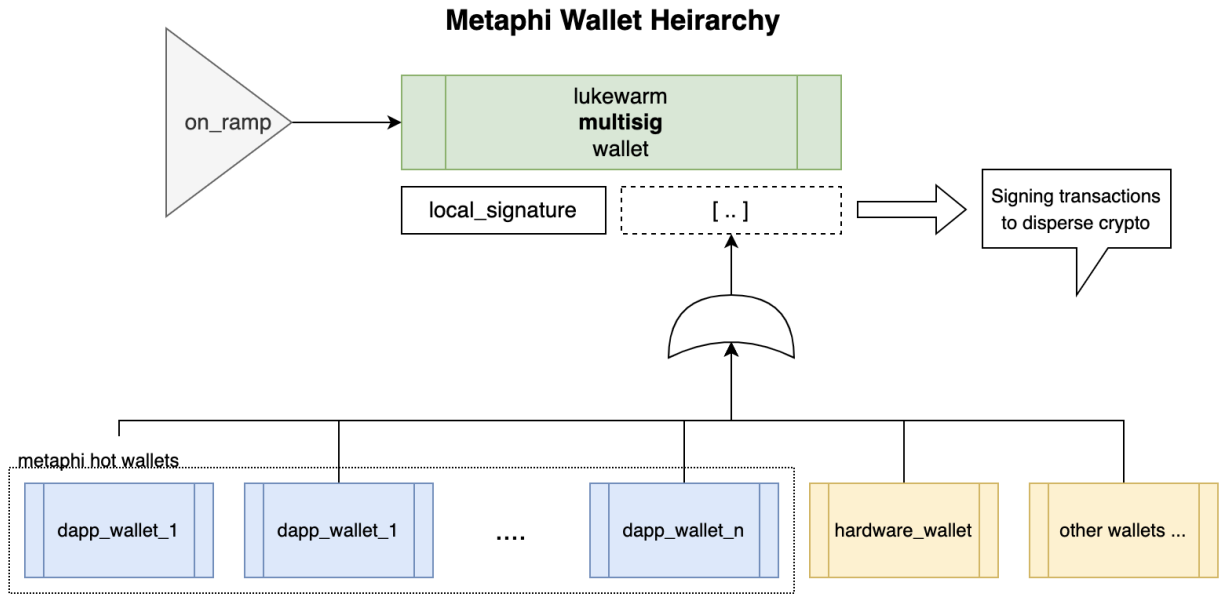
Universal Access & Visualization

Since Metaphi hosts the wallet addresses' and parts of the share for all users, Metaphi is able to provide a visualization of all assets owned, across all dApps to the user directly.

Further along, we can also enable universal signing for the user, from a free-standing Metaphi wallet itself, without a dApp, since the User Device and Metaphi are enough to be able to regenerate the secret keys for all dApps.

Wallet hierarchy

Since our scheme is dApp specific, it has the potential of increasing user cognitive overhead if they interact with multiple dApps and want to either move funds around from one wallet to another or want a unified view of all their wallets across multiple dApps. To address this, we propose a wallet hierarchy, where a top level wallet "owns" several dApp specific wallets. Each of these dApp specific wallets can themselves own multiple sub-wallets, enabling an arbitrarily nested hierarchy. The specifics of how this hierarchy is materialized and managed is an implementation detail we omit for brevity.



Link: Flowchart Maker & Online Diagram Software

Open Source Metaphi SDK

We are building the system described above, into SDKs that can be used by dApps across different platforms.

The core functionality provided by the SDK is:

1. Wallet Creation
    a. Generating a wallet address.
    b. Generating secret shares.
    c. Retrieving users' biometric credential.
    d. Generating symmetric encryption key from retrieved biometric credential.
    e. Encrypting secret shares with this symmetric key.
    f. Uploading secret shares to device, dApp contract and Metaphi.
2. Signing Transactions
    a. Retrieving users' biometric credential.
    b. Generating symmetric encryption key from retrieved biometric credential.
    c. Recreating private key retrieved from device and dApp.
    d. Persisting private key for session duration, and/or based on user setting.

Threat Models

Metaphi database is compromised or unavailable

The secret sharing protocol ensures that a single secret is useless. The secret in the Metaphi Database is also encrypted with a symmetric key generated from the users'

biometric credentials. Also, since only 2 out of 3 keys are required to regenerate the private key, the user will remain unaffected by a breach or an outage.

Metaphi has a malicious insider

Secrets stored in the Metaphi DB are double encrypted with the user generated symmetric key and a cloud provider based KMS solution, backed by a FIPS compliant HSM. So even if the insider manages to break the symmetric key encryption, the Metaphi custodial share remains protected with a key stored in a HSM, ensuring an audit trail for investigation.

User device is stolen

All key segments are encrypted with the users' biometric credentials which will be hard to forge, even if the device is stolen. Even if the biometric guardrails are breached, we have planned a relayer infrastructure to revoke user privileges on demand, which we will detail in a future whitepaper.

For recovery, if the user reports a stolen device, using the two parts on the Metaphi DB and the DApp Authentication contract, new secret shares can be generated on a new device.

1. Device is locked: This secret share is protected by the device protection itself.

2. Device is unlocked

    2.1 Connected to DApp

If the user is connected to the DApp, the private key on the device is open to being compromised. This risk is the same as if a user were to leave their GMail account open. To handle this risk, we have the revoking mechanism using the relayer in our roadmap. The revoking mechanism will work via the DApp Auth Contract and MetaphiDB where all signing credentials from the compromised address will be rejected. The persistence of the generated private key on the device is limited to the session time of connection. This ensures that the attack window is narrowed.

2.2 Not connected to DApp

If the user is not connected to the DApp, the malicious actor would still have access to the encrypted secret share. This is protected by the biometric-based symmetric encryption key.

Phishing Attack for Biometric Credential

If the user is phished for their biometric or pin and the symmetric public key is regenerated by an authorized actor, they would still need to authenticate against Metaphi and the dApp to fetch at least one other share.

Brute force attack

Brute-forcing the pin is made more difficult by the fact that most anticipated use cases for the Metaphi wallet are not expected to last beyond 90 minutes.

DApp is malicious

Metaphi aided wallets have a 1-1 mapping between the DApp and the user. A malicious DApp may only get access to the private key of the wallet that the user is exclusively using for that DApp. Our SDK provides isolation between DApps by only storing the signing provider for the specific DApp.

Impact Analysis

With the dual layers of encryption, spread across three stakeholders, even in the worst case where 2 out of 3 stakeholders are compromised, the impact is always limited to a single user, as described in the table below.

Compromised Secret 1	**Compromised Secret 2**	**Impact**
User Device (all regenerated private keys are compromised)		Single user, all dApp accounts compromised Remediation: Revoking Mechanism from dApps
User Device	Metaphi DB, accessed by	Single user, single dApp account compromised

(biometric key compromised)	network (Whitelisting compromised)	
	Metaphi DB, accessed directly (KMS is compromised) (Whitelisting NA)	Single user, all accounts compromised Impacted users' accounts across single dApp are compromised
dApp Contract	Metaphi DB	Two layers of encryption ● KMS ● Biometric-based encryption
	User Device	● Creates Auditable trail ● Described under *Thread Models: User Device is stolen*

Conclusion

We describe the Metaphi wallet, a non-custodial wallet offering a custodial experience. We described its key core pieces, their interactions and a simple proof demonstrating the properties of our system. We analyzed the threat model and showed that our design is robust against various attacks. We contrasted our system against peer implementations and demonstrated the simplicity of our design. In short, the Metaphi non-custodial wallet solution allows secure management of user's private keys with no impact on their autonomy.