A Decentralized Mnemonic Backup System

PriFi Labs Inc.

Abstract

1 Introduction

"If you do not own your private keys, you do not own your crypto"

This crypto adage encourages crypto enthusiasts to use non-custodial crypto wallets in which the private keys are directly stored on their device. However, using non-custodial wallets comes with the risk of losing all of our crypto assets if we lose our private keys. If our device gets lost, stolen or irremediably break down, our crypto assets might be locked forever [1].

Fortunately, most non-custodial wallets such as Exodus (Bitcoin), Metamask (Ethereum) and Keplr (Secret Network) offer a way to recover our private keys by the mean of a "recovery phrase" also known as "mnemonic seed phrase". It is the 12-word phrase that we are being given the first time we setup our wallet. Here is an example of such a phrase:

```
witch fox practice feed shame open
despair creek road again ice least
```

That phrase is important as it is used to generate the same private keys on demand (BIP39 standard [2]). We can use them as a backup or to import our wallet into a new device. Indeed, this mnemonic phrase is a really sensitive piece of information. Anyone who has access to this phrase would have full control over our crypto assets as explained in the [Metamask FAQ page][3]:

> *"MetaMask requires that you store your Secret Recovery Phrase in a safe place. It is the only way to recover your funds should your device crash or your browser reset. We recommend you to write it down. The most common method is to write your 12-word phrase on a piece of paper and store it safely in a place where only you have access. Note: if you lose your Secret Recovery Phrase, MetaMask cant help you recover your wallet. Never give you Secret Recovery Phrase or your private key(s) to anyone or any site, unless you want them to have full control over your funds."*

As understood here, we are not supposed to remember that phrase like a password. Instead, it is recommended that we write it down on a piece of paper and store it in a "safe place". But where is that "safe place" exactly? In our physical wallet with our cash and credit cards? Not ideal as it could get lost or stolen. Inside our home? No ideal again as it could burn down. A safe deposit box in the vault of a bank? it might be the best solution after all but the logistics that comes with it makes that solution very cumbersome.

In the end, it is very inconvenient to store physical objects safely. So, could we design a simple application that would take custody of my passphrase and would allow me to recover by simply using my email? Yes we could build such an application, it would be something similar to a password manager such as *1password* [4] for instance. However, such a solution requires 1) that the service provider is trustworthy an 2) that the whole application is secured. Indeed, such a centralized solution goes the philosophy of decentralized application.

In this paper, we propose a Decentralized Mnemonic Backup system that anyone can use to give custody to any blockchain passphrase to the Secret Network [5] and recover it using a simple email.

2 The Cryptography Protocol

2.1 Key Registration

Step 1 - Registration Request

1.1 Alice enters her $email$, her passphrase m, an id for her passphrase, the number of shares i she wants to generates and the threshold value j and pressed the submit button

1.2 Then, a script running inside the page generates an ECDH private and public key pair (sec_A, pub_A) and a nonce n_A

1.3 Finally, the script sends the transaction $\mathsf{request}[t_{id}, b_{id}](email, id, pub_A, n_A)$ to the Secret Network using the throwaway Secret account that Alice has just provisioned.

Step 2 - Registration Process

2.1 The Secret Recovery Contract receives the register transaction request and generates a new ECDH private and public key pair (sec_B, pub_B) and a nonce n_B

2.2 Then, calculates the ECDH secret $s = \mathsf{ECDH}(sec_B, pub_B, pub_A)$ and derives the 128-bit AES symmetric key k using the standard password-based key derivation function PBFK2 and the concatenation of n_A and n_B as a salt $k = \mathsf{PBKDF2}(s, n_A||n_B)$

2.3 Finally, the contract stores the record $(email, id, t_{id}, b_{id}, pub_B, n_B)$ in the `register` dataset.

Step 3 - Confirmation Email

3.1 Any transaction sent to the Secret Recovery Contract Contract triggers an event handler on the mailer backend. Once a transaction has been validated, the mailer backend sends a query info() to the Recovery Contract to get information about that transaction.

3.2 The Secret Recovery Contract checks that the query comes from the mailer backend address and retrieves the record from the `register` dataset and checks that the query has not expire based on the initial block id b_{id} and the current block id on the Secret Network

3.3 Then, it returns the *email*, the public key pair pub_B and the nonce n_B.

3.4 Finally, the mailer daemon sends an email to Alice with the confirmation code (pub_B, n_B)

Step 4 - Confirmation Request

4.1 Alice sees a webpage that asks her to enter the confirmation code sent by email. She opens her email and copy'n pastes that confirmation code into her browser.

4.2 Then, a script running inside the page calculates the ECDH secret $s = \mathsf{ECDH}(sec_A, pub_A, pub_B)$ and derives the 128-bit AES symmetric key k using $\mathsf{PBFK2}$ and the concatenation of n_A and n_B as a salt $k = \mathsf{PBKDF2}(s, n_A||n_B)$

4.3 The script calculates a hash-based message authentication code (abbreviated HMAC) using the key k and the concatenation of the email and id $h = \mathsf{HMAC}(k, email||id)$

4.4 Finally, the script sends the transaction $\mathsf{confirm}[t'_{id}, b'_{id}](email, id, h)$ to the Secret Network

Step 5 - Confirmation Process

5.1 The Secret Recovery Contract receives the $\mathsf{confirm}$ transaction and checks that the query has not expire based on the initial block id b_{id} and the current block id on the Secret Network

5.2 Then, it calculates the HMAC of the concatenation of the email and id using the key $h = \mathsf{HMAC}(k, email||id)$ and checks that this hash is strictly equal to the hash h in the request

5.3 Finally, the contract stores the tuple $(email, id, k)$ in the `confirm` dataset.

Step 6 - NFTs Minting

6.1 Once the transaction has been validated, the client-side script can now encrypt (using AES in GCM mode) the passphrase m with the key k to obtain the ciphertext $c = E_{\mathsf{AES}}(k, m)$

6.2 Then, the script concatenates the id and the ciphertext to generate the i shares using the SSS algorithm: $(s_1, \cdots, s_i) = E_{\mathsf{SSS}}(id||c, i, j)$

6.3 Finally, the script mints the i Secret NFTs on the Secret Network

Once the protocol is completed, Alice can now transfer the NFTs to her accointances.

2.2 Key Recovery

Step 1 - Recovery Request

1.1 Alice creates a new Secret Wallet and asks her friend to send her NFTs back. Once she has at least j of those NFTs on her wallet, she goes to the recovery page that asks her to select the NFTs.

1.2 Then, the script in the webpage collects the shares and extracts the id and the ciphertext from the NFTs $id||c = D_{\mathsf{SSS}}(s_1, \cdots, s_j)$

1.3 Finally, the script generates a nonce n_A and sends the transaction $\mathsf{request}[t_{id}, b_{id}](email, id, pub_A, n_A)$ to the Secret Network

Step 2 - Recovery Process

2.1 The Secret Recovery Contract receives the transaction and retrieves the record from the `confirm` dataset using the email and id from the transaction

2.2 Then, the script generates a nonce n_B and stores the record $(email, id, t_{id}, b_{id}, n_B)$ in the `register` dataset.

Step 3 - Verification Email

3.1 Any transaction sent to the Secret Recovery Contract Contract triggers an event handler on the mailer backend. Once a transaction has been validated, the mailer backend sends a query $\mathsf{info}()$ to the Recovery Contract to get information about that transaction.

3.2 The Secret Recovery Contract checks that the query comes the mailer backend and retrieves the record from the `register` dataset and checks that the query has not expire based on the initial block id b_{id} and the current block id on the Secret Network

3.3 Then, it returns the *email*, the public key the nonce n_B.

3.4 Finally, the mailer daemon sends an email to Alice with the verification code n_B

Step 4 - Verification Request

4.1 Alice sees a webpage that asks her to enter the verification code sent by email. She opens her email and copy'n pastes that validation code into her browser.

4.2 Then, the script calculates the hash of the concatenation of the two nonces $h = \mathsf{H}_{\mathsf{SHA512}}(n_A||n_B)$

4.3 Finally, the script sends a query $\mathsf{verify}[t'_{id}, b'_{id}](email, id, h)$ to the Secret Network

Step 5 - Recovery Process

5.1 The Secret Recovery Contract receives the verify transaction and checks that the query has not expire based on the initial block id b_{id} and the current block id on the Secret Network

5.2 Then, it calculates the hash of the concatenation of the two nonces $h = \mathsf{H}_{\mathsf{SHA512}}(n_A||n_B)$ and checks that this hash is strictly equal to the hash h in the request

5.3 Finally, it returns the key k back to the client

Step 6 - Passphrase Decryption

6.1 After receivving the key back, the client-side script can now decrypt the ciphertext c to obtain the passphrase $m = D_{\mathsf{AES}}(k, c)$

Once the protocol is completed, Alice sees her passphrase on the webpage.

3 Discussion

This backup system will demonstrate that the Secret Network privacy model is adequate to keep sensitive information confidential. Key features of Secret NFTs: secret metada and private ownership

Why implementing our backup system on Secret Network rather than Ethereum? Is it because Ethereum-based NFTs do not have secret metadata? Not really since we could have created a password protected version of the share and hid it inside a public NFT image using Steganography [6]. The real problem is that if an attacker knows that an NFT contains a share to recover a mnemonic phrase, he or she will be able to locate all others NFTs that contains the other shares easily since ownership is public on Ethereum. The fact that Secret NFTs protects the ownership is the key feature that makes our backup system secured.

4 Conclusion and Future Work

The key recovering system can be used outside of our Mnemonic Backup system. It can be used for more advanced cryptographic protocols that involve storing a secret key on the blockchain with the option of recovering it using an email.

References

[1] Lost Passwords Lock Millionaires Out of Their Bitcoin Fortunes, https://www.nytimes.com/2021/01/12/technology/bitcoin-passwords-wallets-fortunes.html

[2] Mnemonic Generation (BIP39) Simply Explained, https://medium.com/coinmonks/mnemonic-generation-bip39-simply-explained-e9ac18db9477

[3] Metamask FAQ page, https://metamask.io/faqs/

[4] 1password, https://support.1password.com/forgot-account-password/

[5] Secret Network: A Privacy-Preserving Secret Contract & Decentralized Application Platform, `https://scrt.network/graypaper`

[6] Steganography https://en.wikipedia.org/wiki/Steganography)