SMS Wallet

By Aaron Li, July 11, 2022 (Presentation: SMS Wallet)

Overview

The SMS Wallet is a lightweight non-custodial wallet solution for users to quickly create a wallet bound to their phone number, receive (or purchase) inexpensive NFTs, showcase their NFTs, and to hold a small amount of crypto assets. The wallet primarily operates on a mobile browser, and is recoverable using text message verification codes in combination with a QR code saved by the user during the wallet's creation.

Once created, some operations for the wallet may also be completed by text message commands. These operations allow NFTs to be minted on or transferred to the user's SMS wallet. The wallet may also allow the user to give the operator (a server) the custody over some assets for specific purposes, such as transferring native assets or NFTs to specific addresses. This expands the scope of actions that can be completed solely with text messages commands.

The SMS Wallet may also allow a user to directly purchase NFTs using credit cards, or claim NFTs for free in some instances. During this process, some native assets (ONE) may also be sent to the user's wallet as gifts and to provide sufficient gas fees for future transactions. The SMS Wallet may also allow purchase of native assets using credit cards. However, such purchases should be exclusively handled in pop-up widgets by low-friction fiat-gateways (Transak, Simplex) with money-transmitter licenses, and the users must be promptly notified so.

The SMS Wallet also provides the user a way to off-board when the user accumulates a meaningful amount of assets. In one click, the user may transfer all assets to a standard private-key based wallet such as MetaMask. The user will also be provided with an option to create a smart contract wallet (such as 1wallet) following a usual onboarding process, and transfer all assets there in one-click.

Description

The SMS Wallet is entirely focused on users, not creators. It aims for simplicity and ease of use, and presents a very opinionated, limited set of features to the user. It should be distinguished from creator-oriented wallets which would focus on features such as sales management, NFT contract administration, and asset administration. Although the creator-oriented wallet may share some core security designs from the SMS Wallet, it should be considered as an entirely separate product.

The SMS Wallet can be broken down into the following product features:

1. Core Security Features
    a. Creating a wallet
    b. Restoring a wallet
2. NFT acquisition (scan QR code / visit direct link)

    a. Claiming free NFTs
    b. (Optional, High Priority) Purchase NFT using credit cards (Stripe / Paypal)
3. (Optional, Low Priority) SMS controlled mini-wallet
    a. Transferring native assets up to a certain amount to specific addresses
    b. Managing NFT assets
4. Fiat gateway widgets for purchasing native assets
    a. (Optional, Low Priority) Simplex
    b. (Optional, Medium Priority) Transak
5. Off-boarding
    a. (Optional, Medium Priority) To standard private-key based wallet
    b. (Optional, Low Priority) To 1wallet

Technical Description

Based on an initial survey, we did not find any comparable non-custodial wallets that make meaningful uses of phone numbers or text messages. For the purpose of technical analysis, one product that is relevant here is Argent, which makes use of phone numbers to verify a user is a real human, to send alerts, and to use text messages as a second factor in 2-factor authentication for wallet recovery.[1] However, there is little documentation available explaining how Argent designs and implements the mechanisms behind the scene, and what Argent did to ensure the security of this process.

Here, we discuss each product feature separately and provide breakdowns and some analyses on technical components.

Core Security Features

Given the relatively insignificant amount of assets in the user's wallet, the mobile-browser runtime environment, we assume the client is relatively secure and immutable[2], such that it is safe to store the wallet's private key as volatile, first-party cookies[3].

However, we must do our best to ensure the plain text private key never leaves the mobile-browser environment. We must not ask users to make any backup material (such as seed phrases) that may derive the private key on its own, nor shall we allow any server to derive the private key either solely using the information available at the server.

Yet we still want the user to be able to fuse the pieces and restore the original private key after verifying the user owns the same phone number that created the wallet. The following sections explain how we can do this while maintaining reasonable security.

For the purposes of this section, we follow these notations:

---

[1] See https://support.argent.xyz/hc/en-us/articles/360025042551-Phone-number-verification-privacy
[2] In implementations, we need to use iframe embedding and IPFS to ensure that. See discussions later
[3] Note that on iOS, the cookies are automatically deleted after 7-days of non-usage

n - user's phone number
z - server secret (transient), 256 bit
s - client generated private key, 256 bit
p - client generated encryption secret, 256 bit
h - the keccak256 hash function
enc - a symmetrical encryption algorithm, e.g. AES-CBC with p as initial vector. Here, enc(a, b) means to encrypt b using secret a,
dec - the corresponding symmetrical decryption algorithm. Here, dec(a, b) means to decrypt b using secret a
otp - the TOTP algorithm, otp(a, t) means to generate the TOTP for time t using a as the secret

The technical components here are:

1. a standard REST API server, e.g. in node.js
2. a private database that is accessible only by the server, e.g. GCP Datastore
3. a web client, e.g. written in Javascript with React
4. a client host, ideally immutable, e.g on IPFS[4]

Create

1. user enters n
2. locally generate s, p
3. set q = h(p || n)
4. user save p as QR code photo
5. compute e = enc(p, s)
6. client sends q, e, n to server
7. server caches (n, e, q)
8. server texts otp(q || z, t) to n
9. user receives otp' and client sends otp' to server
10. server checks if otp(q || z, t) == otp' and persists (n, e, q)
11. client erases p, e from memory
12. client persists s in cookies (expires in 7 days when unused)

Restore

1. user scan/load p (QR code photo)
2. user enters n
3. set q = h(p || n)
4. user sends (n, q) to server
5. server verifies (n, q) exists and load (n, e, q)
6. server texts otp(q || z, t) to n
7. user receives otp' and client sends otp' to server

---

[4] In practice, we will assign an easy-to-remember domain name to host the wallet. To prevent the hacker from hijacking the domain, loading a fake client, and stealing the private key in the cookies, we may load the client in an iframe that points to a fully resolved IPFS domain, so cookies are only stored under such IPFS domain. The hacker's fake client would have a different IPFS domain (due to content change) hence unable to find any cookies under there.

8. server checks if otp(q || z, t) == otp' and sends e to client
9. client computes s = dec(p, e)
10. client erases p, e from memory
11. client persists s in cookies

Security Analysis

1. Assume the server is not compromised
    a. If the user's saved QR code (photo) is stolen, the hacker cannot obtain the user's private key without getting a copy of the user's encrypted private key e.
    b. The hacker cannot obtain the user's encrypted private key e unless the hacker also compromises the user's phone or be able to intercept the user's text messages
2. If the server is compromised, the encrypted private keys e for all users are obtained by the hacker
    a. The hacker cannot obtain any user's private key without first obtaining the user's encryption secret p, which is stored locally as a photo in the user's device
    b. The hacker cannot obtain p by intercepting traffic at the server or sending malicious responses to the user
    c. The only way for the hacker to obtain p (other than to steal from the user directly) is to also compromise the client (e.g. via phishing) and intercept p when some users want to restore their wallet. To reduce such risk, the client should be served by isolated infrastructure (e.g. IPFS or others), and not be served using the same server. See also the footnote in the last page.

NFT Acquisition

NFT acquisition can be implemented with or without the involvement of a server. The approach without a server requires us (the operator) to provide a small amount of native assets upfront to newly created wallets as gas fees for client-initiated transactions. This approach is more “decentralized”[5], but has many implicit limitations. For example, we cannot easily rate-limit the number of claims a wallet might make across NFTs from multiple contracts. In the case of supporting credit card purchases, a server has to be used to reliably verify the status of a credit card transaction before a (paid) NFT can be delivered to a user. For these reasons, a server based implementation is more advantageous in terms of simplicity, functionality, and reusability.

Sending NFT

From the view of a server, the process of serving a request to claim free NFTs is not different from the process of honoring a NFT purchase via credit card[6]. Once the server confirms which

[5] in the sense that it distributes the workload to nodes in the blockchain network
[6] except the latter requires the server to first receive a confirmation (from fiat payment gateway) that the credit card purchase is completed, and to identify which NFT is being purchased

NFT is to be sent to the user, all that remains is to deliver the NFT to the user's wallet. The most common delivery methods are:

1. to execute a (custom) **mint** function on the NFT contract with a destination address set to the user's address, or
2. to execute **safeTransferFrom**, the standard transfer function of ERC721 or ERC1155 that moves the ownership of the pre-minted NFTs to the user.

Note that, (1) is a custom function that does not exist in any NFT standard, whereas the transfer function in (2) is standardized. The actual mint function implementations vary between projects[7]. The benefit of (1) over (2) is the NFT (and its metadata) may be concealed until it is minted, whereas in (2) the user may be able to view the pool of NFTs available (and their metadata), and potentially predict which NFT they will get. In practice, the difference is negligible to average users especially for low-value NFTs and for claiming processes with little randomizations. For simplicity, approach (2) is recommended. We may implement approach (1) when we receive feedback from users or creators that approach (1) cannot meet their needs.

Preparations

For implementation, a separate server[8] with a dedicated private key (and wallet address) can be used to facilitate the transfers when users claim or purchase NFTs. The following preparations should be done prior to enabling NFT claims or purchases for each NFT contract. Here, we refer the NFT contracts' admin(s) as Creators:

1. Creators should mint some NFTs ahead of time, and specify which ones are claimable (as free NFTs), and which ones are purchasable and at what price
2. Creators should register the NFT contracts, token IDs, and fiat prices with the server
3. Creators should transfer the minted NFTs to a separate wallet address ("Vault") controlled by the Creators
4. Creators should approve the server's wallet address to operate the tokens in the Vault (use **setApprovalForAll** or **approve** functions in the standard)

When the server processes a request for claiming free NFTs, the server should verify the user's phone number (using SMS and OTP) before proceeding with executing the transfer function on the NFT contract.

The server should integrate Stripe or Paypal credit card purchase APIs and callbacks. When the server processes a purchase, it would wait for the appropriate callback from Stripe or Paypal to trigger, before looking up the purchased NFT, and the user's phone number and wallet information before sending the NFT to the user using the same transfer function.

Security Analysis

This setup ensures in the worst case that the server's private key is completely compromised, the only NFTs at risk would be the ones transferred to the Vault which the creators explicitly

[7] We may include a mint function as part of a contract template for the creator's version of the wallet.
[8] different from the one providing core security features

approved the server to control. Usually, this only includes a small amount of NFTs in a single round of sale. This setup effectively prevents catastrophic incidents when a creator's entire collections get stolen by hackers, or to have the entire contract under control by the hacker[9].

SMS Controlled Mini-wallet

Broadly speaking, there are two types of SMS commands concerning the user's wallet:

1. Commands that do not require the user's authorization, for example, claiming an NFT. These commands can be completed by the server initiating a blockchain transaction for the benefit of the user.
2. Commands that require the user's express authorization, for example, transferring X amount of native asset to address Y, transferring an NFT A to address Y, approving address Y to operate NFT. Usually, the express authorization is manifested by the user "signing the transaction", i.e. using the private key to encode a JSON RPC request with transaction data.

For type (1) SMS commands, the server can simply execute the required transaction using its own private key, after the server receives a callback from SMS gateways (such as Twilio) that contains the user's SMS commands.

Serving type (2) SMS commands requires a more complex processing system. Here, any express authorization from the user can only be given by text messages in English, and the user's private key is unobtainable throughout the process[10]. Therefore, to achieve this functionality, it is inevitable that the server gains custody of some of the user's assets. To minimize security risks, the scope and size of operations the server is permitted to perform with respect to any user's asset should be limited and configured by the users themselves.

One way to achieve this goal is through a smart contract with the following functions:

1. **deposit**: allows the user to send some native assets to the smart contract, subject to a system-wide limit[11].
2. **withdraw(uint256 amount)**: allows the user to withdraw native assets the user previously deposited into the contract.
3. **authorize(uint256 limit, address dest)**: allows the user pre-authorize cumulatively up to **limit** amount of native asset to be sent to **dest** by the operator, so that the operator may do so when it receives such SMS commands from the user
4. **send(uint256 amount, address from, address to)**: allows the operator to send a certain **amount** of native tokens deposited by user **from** to address **to**, provided that

---

[9] e.g. the hacker may mint a large quantity of valuable NFTs for themselves

[10] Since we want to keep the interactions as simple as possible, we should not ask the user to open their browser. For security reasons, we should not ask the user to provide their encryption secret either, or transmit it to the server in any way.

[11] The server's wallet (i.e. the operator) should not be allowed to adjust this limit. Only the admin (the developer managing the contracts) should be allowed to change this configuration.

user **from** has already authorized more than such **amount** of native tokens and the limit is not used up.

5. **transfer(uint256 amount, uint256 tokenId, address contract, address from, address to)**: similar to **send**, but works for ERC20/721/1155 tokens. Note that approval functions (equivalent to pre-authorization) of the tokens reside on the token-contracts themselves, not this smart contract.

The contract may also maintain some read-only functions and public variables to allow pre-authorization and deposit information to be looked up by the users and the operator.

Fiat gateway widgets

See polymorpher/1wallet implementation and 1wallet.crazy.one “Buy” button experience.

Off-boarding

Transitioning to MetaMask

If the user is on desktop, the browser page could guide the user to install MetaMask then invoke **eth_requestAccounts** on the web3 provider to obtain the user’s MetaMask address. After that, the client could submit multiple transactions that transfer all assets to the MetaMask address, pending the user’s final confirmation. The whole process can be completed in two clicks on the web page (one click to connect MetaMask, one click to provide confirmation), plus a few clicks on the MetaMask extension.

If the user is on mobile, the browser page could use a deep link that automatically opens MetaMask mobile app and loads a custom, unique URL in the in-app browser. The script at the URL will capture the user’s MetaMask wallet address and rewrite the URL, then instruct the user to open the current page in-browser (using the "…” button on the bottom right). After the user opens the page in the browser, the transfer of assets can continue in the same way compared to the desktop case.

Transitioning to Other Private Key Wallets

If the user is on desktop, the browser page could show a wallet-connect QR code to begin a wallet-connect session. Most private-key wallets support wallet-connect. After the session is established, the browser page could obtain the user’s destination wallet address and continue the same way as described in MetaMask desktop case.

Transitioning to 1wallet

A dedicated endpoint can be built in 1wallet web to accommodate the transition. The user would be guided to create a 1wallet (or choose an existing one if the user has any). Afterwards, the user would be redirected back to the standard off-boarding page (similar to MetaMask desktop case) where the user can provide the final confirmation of transferring all assets

Other Notes

Generalized Authentication

As discussed previously, the Core Security Features may also be used for a wallet which primarily uses email for authentication. In that case, the verification codes can simply be sent to email addresses instead of phone numbers, and the encrypted key of a user is bound to an email address. However, a few key differences should be noted:

1. Since email is much easier and much cheaper to obtain than a phone number, the risk of bots and spam becomes significantly higher. Certain strategies that make sense for users who verified their phone numbers may become problematic for users who only verified their email addresses. For example, the proposal that allows users to claim free NFTs, or the one that provides a small amount of native assets for newly registered users. Email-only verification makes it very easy for someone to abuse the system as long as there is an incentive to do so.
2. Email is not as interactive as text. In fact, it is almost unheard of that any user would expect to reply back to an email to confirm a transaction, or to send a command to a certain email address to initiate any action. Most users expect emails to be read and processed by humans. Therefore, we cannot simply replace “SMS Controlled Mini-wallet” by “Email Controlled Mini-wallet”. The mini-wallet design might not work at all in the case of email, despite the fact that we have the ability to generate a unique recipient email address for each user. There is an inherent risk of sender-email address spoofing, and people may write emails in many different languages, formats, encoding, or signatures.
3. Email is a great tool to display rich content and to follow-up with customer support conversations, subscription content, or interesting statistics. SMS cannot do that.

Signing Arbitrary Transactions

SMS Wallet may be used to sign arbitrary transactions as well. Consider a game that needs a SMS Wallet user to approve a smart contract transaction. The game could call an API which SMS Wallet backend may provide. The backend would create a compressed, unique URL[12] mapped to the user’s wallet address and the transaction, then text the URL to the user’s phone number. The user may click the URL in a text message, and confirm signing in the browser (which has access to the wallet’s private key as the URL would share the same domain).

Cross-device Usage

SMS Wallet can be used simultaneously and independently on both the user’s phone and computer. The user may simply restore the wallet on a desktop computer after they created it on the phone. The usage flows are identical.

[12] on the wallet’s domain where the private key cookies reside

Security Options

We may allow a user to “logout” from the browser at any time. The client would delete all the cookies, hence the private key cached in the browser. Similarly, we may call “restore” as “login” instead. For extra security, we have the option to reduce cookies expiry time, or not to store the private key in the cookies at all. In that case, we would require the user to retrieve the encrypted private key from the server every time the user wants to make any transaction, hence forcing the user to verify the phone number with an OTP code every time, essentially turning the SMS Wallet as a 2FA wallet using SMS as the secondary factor for security.