Open Beta Launch Whitepaper (27 July 2021)

Introduction

Harpie is keyless loss prevention for your Ethereum tokens. If you ever lose access to your digital wallet, Harpie retrieves tokens out of your lost wallet and moves them to a new one. We never see or store your private key.

Our mission is to provide all crypto investors, regardless of holding size, with a reliable source of peace-of-mind. We're wallet-adjacent, meaning that you'll be able to keep using your hardware or software wallet and still sign up with Harpie. We also never hold your crypto in custody, meaning that the crypto you protect is still liquid.

We're secured under a zero-knowledge proof that ensures the following:

- We never hold or see your private key.
- We can never access your wallet.
- Fewer cyber-attack vectors.
- A 100% transfer success rate after security checks are completed.

Find peace of mind for your crypto holdings. Sign up for Harpie today.

Table of Contents

I: Mission

Accessible to all, not just corporations.

Current redundancy solutions are restricted to insurance and key custody: both of these solutions cost a significant amount of money to upkeep. Harpie's tech is low-overhead, meaning that we don't have to charge high prices.

Security-centered development.

We have a minimized tech stack with local dependencies, we host on Microsoft Azure's servers, and we use military-grade encryption schemes that never require a private key. This helps protect against a multitude of attack vectors. See section III.

Simple registration.

Tech know-how shouldn't be a barrier to using Harpie. Anyone that has an Ethereum wallet outside of an exchange can use Harpie without any hiccups.

Completely transparent tech.

All of our code is publicly available. We run coding bounties to ensure global collaboration on key features; this ensures that development isn't completely centralized.

For the people.

One of our goals for open-source development is enabling developers to run local nodes of Harpie for free. This allows anyone to take complete charge of their asset security while using Harpie as a tool to do so.

Current Issues

Storage Anxiety

A large number of people take great lengths and lose a significant portion of their returns simply to avoid handling their crypto custody. Anxious crypto users will send their keys to friends, keep their crypto in exchanges with limited liquidity and security, or even invest in crypto ETFs with high fees.

Lack of Transparency

Some crypto insurance solutions in the market do not have open-source code, and some even avoid transparency about their holdings. This poses a huge risk if a company goes under, and could even contribute to companies going under in the first place.

Unsafe Industry Practices

A large number of crypto companies abuse the “approval” function of Ethereum tokens: when you sign up for these services, you essentially entrust the entirety of your token holdings to that company. It breaks the idea of trustless transactions.

Expensive and Inconvenient Custody

Average consumers (AKA: the group that blockchains should be helping the most) cannot afford the prices that premium custody solutions charge. In addition, these custody solutions don’t allow you to move your money out quickly, meaning that you’re paying mountains for your crypto to be stuck in bureaucratic chains.

How Harpie Helps

Possible Applications

- Back up your crypto in case of key loss.
- Have a redundancy plan in place for your hardware wallet.
- Keep your crypto safe even in the case of natural disaster.
- Avoid “trusted-network” wallet solutions.
- Keep essential development tokens safe as you work on them on the mainnet.
- Avoid annoying and unintuitive multi-sig applications.

Trustless (and Trustworthy) Peace of Mind

- Our zero-knowledge proof is the backbone of our service. It ensures that we can never access your wallet with the data we store.
- All of our transactions are visible on the blockchain.
- Your machine locally encrypts the information Harpie collects on user signup. This means that the data that we store consists of randomly-generated numbers and unreadable ciphertexts.
- The only personal info we require is an email address: stay anonymous if you wish.
- Structure-wise: we’re a U.S based corporation incorporated in Delaware (Harpie Blockchain Solutions, Inc). We must follow legal regulations or face severe repercussions.

II: Tech Stack

Transparency is our guarantee.

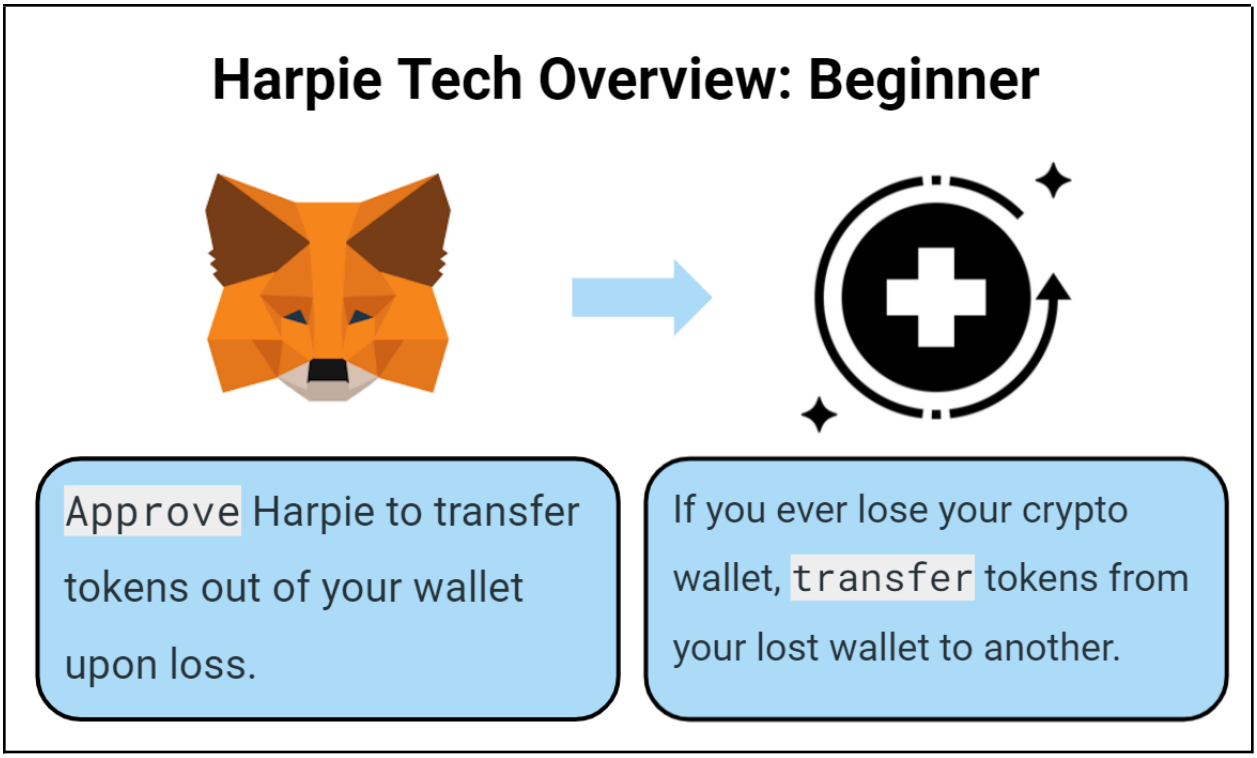
The Harpie team built Harpie because we were fed up with custody/wallet solutions hosting everything on centralized servers, with barely any transparency on their liquidity or their stored information.

We're here to change things up: this tech overview will show all the data we store for each user, the tech that runs Harpie's token protection and recovery service, and the security features we have implemented.

We'll be going over each layer of the tech in levels of abstraction; from beginner, to intermediate, to advanced. To see all the tech in action, see our publicly accessible Git repo as well.

Keep in mind that the Harpie project is still in its infantile stages. We hope that a global community of developers will help us create useful features and strengthen the security around Harpie.

Complexity Level One: Introduction

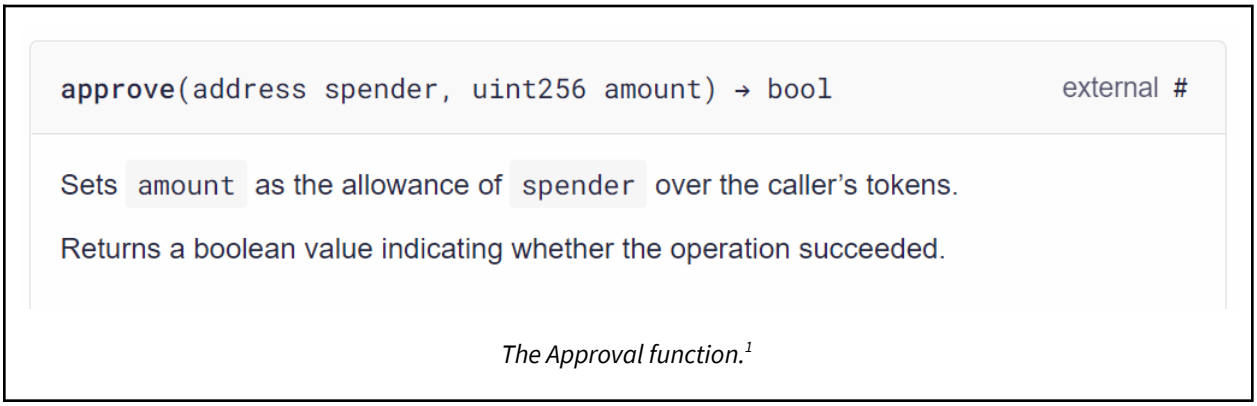


At its very barebones, using Harpie is a two step process.

You'll first give Harpie permission to transfer tokens out of your wallet at a later time. For most of our users, we hope that "later time" never comes. But, for our users that do lose their crypto wallets, we leverage this permission to transfer tokens out of their lost wallets and into their new ones.

This permission is granted with the `Approve` function native to almost all Ethereum token contracts (i.e. USD Coin, DAI, UNI, etc).

The Approve Function



The Approval function.[1]

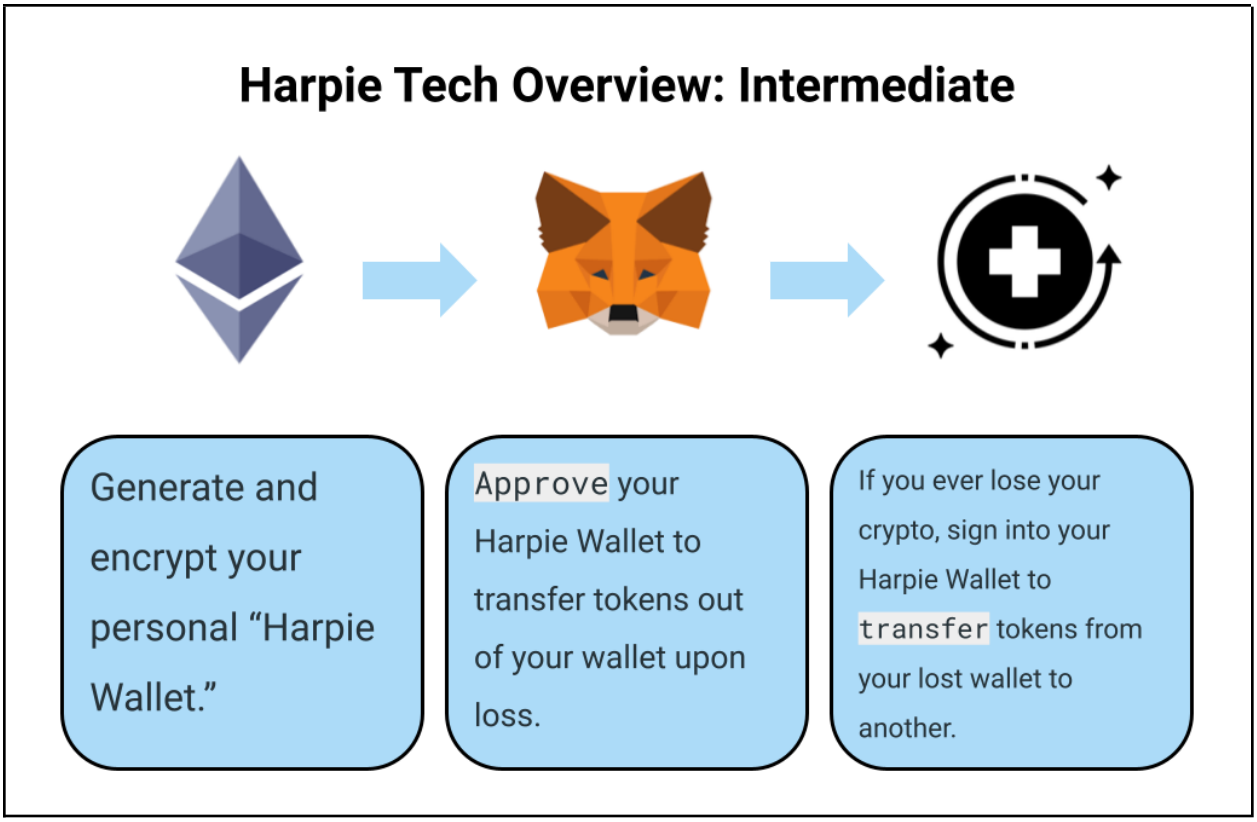
`Approve` has two parameters, `spender` and `amount`. When you `approve` a wallet, you're allowing a `spender` to transfer a certain `amount` of your tokens at a later time. For example, if I `approve` my friend for an `amount` of 500 USD Coin, they can `transfer` 500 USDC out of my wallet at a future time.

So, when a user signs up with Harpie, they're calling `approve`, where the `spender` is Harpie, and the `amount` is equal to the amount of tokens they want to protect.

Note: until a `transfer` is conducted, no exchange of funds will occur. This makes Harpie inherently non-custodial.

[1] ERC-20 Docs. https://docs.openzeppelin.com/contracts/3.x/api/token/erc20#IERC20

Complexity Level Two: Turning Approvals Trustless



Granting a service approval is normally a trust-giving process: once given, you must trust that service to never leverage your approval maliciously (i.e. transferring tokens without your express permission). But Harpie uses a zero-knowledge system to take trust out of the equation altogether.

Zero-knowledge Systems Explained

The idea of a zero-knowledge system is traditionally told with the analogy of a hotel.[2] When you enter a hotel, you have your own card, but the hotel also holds a backup card to each room in storage. If the hotel responsibly manages your backup card, you're secure—but, if the hotel's storage is compromised, your security is also compromised.

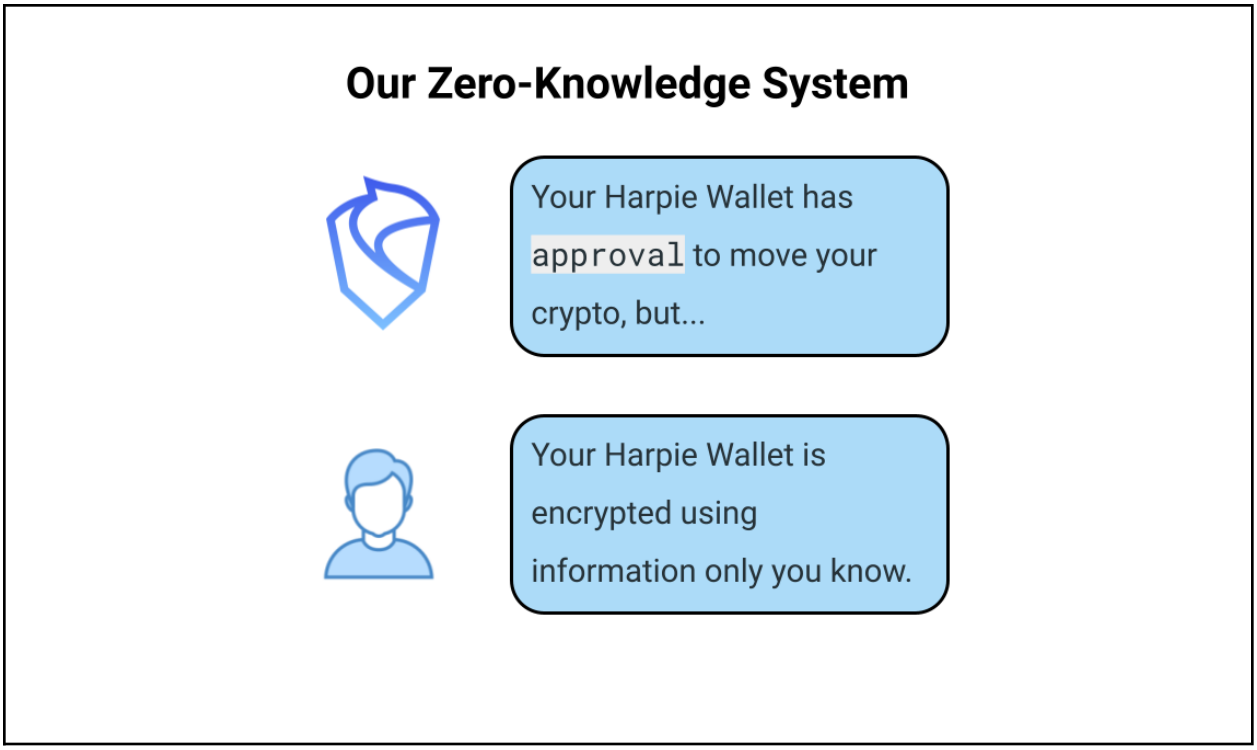
[2] Tresorit: https://tresorit.com/blog/zero-knowledge-encryption/

Now, imagine a similar hotel but with just one difference—there is no storage of cards. This is an example of a zero-knowledge system. The hotel still is able to validate that you own a card, but they have *zero-knowledge* on where your card is.

If your hotel keeps a copy of your card, you need to trust that they store it responsibly. **But with a zero-knowledge system, your security isn't a matter of trust—it's a guarantee.**

When you create your Harpie account, you'll locally generate a unique "Harpie Wallet." This Harpie Wallet's private key is locally encrypted using **information only you know**, and that hidden information is never sent to our servers. Everything is done locally.

When you `approve` Harpie, you're actually approving your secure Harpie Wallet. Because you've locally encrypted the private key to your Harpie Wallet, Harpie will never have access to it. At a later time, if you need to conduct a recovery, you'll unlock your Harpie Wallet and transfer funds on your own. This essentially creates a zero-knowledge system: **Your Harpie Wallet can transfer money out of your wallet, but only you can access your Harpie Wallet.**



Wallet Generation and Encryption

As mentioned earlier, your Harpie Wallet is encrypted locally. At a high level, this is how that process works.

1) After a user signs up with Harpie, they create and answer three security questions.
2) On the browser, the answers to the security questions are combined using a hashing function.
3) On the browser, an Ethereum wallet is generated with a random private key. We will call this the “Harpie Wallet.”
4) On the browser, the private key to the user’s Harpie Wallet is encrypted using the hashed security questions.
5) The following data is sent to our servers: encrypted private key to the user’s Harpie Wallet, public address to the user’s Harpie Wallet, essential encryption information (salts, initialization vectors, etc), and security questions (but not the answers to the security questions).
6) When a user protects their crypto with Harpie, they will `Approve` their Harpie Wallet with their own wallet.
7) Upon recovery, the user will unlock access to their Harpie Wallet using the answers to their security questions. To conduct a recovery, they’ll use the `transferFrom` function on their Harpie wallet to move tokens out of their lost/inaccessible wallet and into a new one. More info on this in the next section.

With the entire picture covered, here are more specifics on the exact algorithms used. All of these computations are done *browser-side*.

1) After a user signs up with Harpie, they create and answer three security questions.
2) The security *answers* are concatenated, then a random salt is concatenated on top. We then use the SHA3 algorithm (or, more accurately, Keccak[c=2d])[3] to combine this full string into a 256-bit hash.

[3] CryptoJS: https://cryptojs.gitbook.io/docs/#hashing

a) The hashing algorithm is provided by the Crypto-JS library, and the salt is randomly produced by the browser's native crypto module, which varies among browsers. Generally, these randomness sources are more secure than JavaScript's `Math.random()`.
b) The salt is in place to further increase the complexity of our encryption and prevent common attack vectors, such as Rainbow table attacks.[4]
c) Note: The next iteration of Harpie will deprecate SHA3 and instead introduce PBKDF-2 as a method to turn security answers into a 256-bit key.

3) We use Ethers.js "`ethers.Wallet.CreateRandom()`" function to create a new Harpie Wallet with a random private key.
4) The private key returned by Ethers is encrypted using the AES-CBC algorithm in 256-bit mode.
    a) This algorithm is provided by the Crypto-JS library.
    b) The initialization vector is randomly produced by the browser's native crypto module, which varies among browsers.
    c) The encrypted private key is virtually impenetrable to brute force attacks.[5]
5) The following data is sent to our servers: encrypted private key to the Harpie Wallet, public address to the Harpie Wallet, essential encryption information (salts, initialization vectors, etc), security questions (but not the answers to the security questions).

Since the wallet creation and encryption process is done locally, Harpie has no way of knowing its own private key before it gets encrypted. Without the private key to Harpie's wallet, there is no way Harpie can leverage its approval to move your tokens.

[4] CAPEC-55: Rainbow Table Password Cracking. https://capec.mitre.org/data/definitions/55.html

[5] NIST: AES https://www.nist.gov/publications/advanced-encryption-standard-aes

Wrapping it Up: Harpie's Zero-Knowledge Proof

It's not enough to simply *claim* that a system is zero-knowledge; it's essential to *prove* it. This is done by conducting an aptly-named zero-knowledge proof.

In a ZKP, a person (a prover) must prove to another person (a verifier) that they know some password, without actually revealing that password. In the example above, the prover is you, who is acting as the hotel guest, the verifier is the hotel, and the password is your room card.

A zero-knowledge proof must prove three things about a system:

1) **Completeness:** if the prover provides a correct password, a verifier will be convinced that the password is correct. (a correct hotel card can be used to enter a room)
2) **Soundness:** the verifier can only be convinced if the prover is providing the correct password (an incorrect hotel card cannot be used to enter a room)
3) **Zero-knowledge:** the verifier must not know, see, or store the password (the hotel does not hold cards in storage).

In Harpie's zero-knowledge system, your security questions act as the "key" to recovering your funds. We satisfy all three categories of a zero-knowledge proof.

1) **Completeness:** If the user provides the correct security answers, they can connect to the Harpie Wallet. This is because web3 cannot connect to the Harpie Wallet without the correct encryption key.
2) **Soundness**: If a user provides incorrect security answers, they cannot connect to the Harpie wallet.
    a) In any zero-knowledge Proof, a "soundness error" exists—this is equal to the probability that an unintended verifier (most likely a brute-force hacker) can unlock the Harpie Wallet. For Harpie's ZKP, this soundness error is extremely, extremely minimal. More concrete numbers are found in the next section.

3) **Zero-knowledge:** Because Harpie never sees your security question answers at any step of the process, we are confident in saying that Harpie is **secured by a zero-knowledge proof.**

ZKP Soundness Error

Because we use SHA3 hashing (the same hash algorithm that Ethereum uses for its keys) to produce a 256-bit key, the chance of a soundness error (a brute-force attack) is equal to the collision probability. This is because the chance of a brute-force preimage or second-preimage attack is lower than the chance of a brute-force collision attack.

SHA3 has 128-bit collision resistance[6]. In other words, it takes a non-trivial (unfeasible) amount of time to create an attack for one of the hashes. To put this in perspective, the entire Ethereum blockchain processes 550 trillion hashes per second as of July 27, 2021[7], or about 2^74 hashes per year. This means that it takes more than 2^40 years to crack a single hash with the combined power of all Ethereum miners.

[6] Gov Info: https://www.govinfo.gov/content/pkg/GOVPUB-C13-939df0873ba823794e0fae5be2fdf28e/pdf/GOVPUB-C13-939df0873ba823794e0fae5be2fdf28e.pdf

[7] Ethereum Network Hash Rate: https://ycharts.com/indicators/ethereum_network_hash_rate

Token Recovery

Harpie's recovery step stays true to our philosophy of using a zero-knowledge proof. Even when Harpie helps a customer conduct a recovery, we **never, ever see or store their security question answers.** Instead, we give the customer the guidance and tools necessary to seamlessly conduct recoveries themselves.

Connection Phase

Harpie provides the user with their essential encryption information (ciphertext, initialization vector, salt, etc) and prompts the user to enter the answers to their security questions. The security questions are then used to decrypt the ciphertext. If the security questions are answered correctly, the user now has access to their Harpie Wallet through the web3 API. If the questions are answered incorrectly, the user must try again.

Transfer Phase

Once the user is connected to their Harpie Wallet through web3, they now have the ability to interact with contracts from the Harpie Wallet. The customer now has the ability to transfer the previously-approved tokens out of the lost wallet to another. Harpie makes this easy to do with a simple button-based interface, but a developer-level user can take the unlocked private key of the Harpie Wallet to deal with the recovery process themselves.

All of the calculations and connections outlined in the two sections above are done browser-side: nothing the user inputs in the recovery phase is ever sent to the Harpie database.

Storage of Your Data

The only personally-identifying information Harpie takes about you is your email address. For our many users that prioritize anonymity: make sure you use an email that doesn't include personally-identifying information in the address!

Here are the contents of Harpie's server:

- Email
- Your public addresses
- Your security **questions** (not the answers)
- Harpie's wallet information, including the public address and securely encrypted private address
- Entropy used to encrypt Harpie's wallet, including salt and IV
- Miscellaneous auditing data
- Necessary login details provided by ASP.NET login APIs

Harpie Server Contents

Email	Your Public Address	Harpie's Public Address	Encrypted Private Key	Salt	IV	Misc.
example@harpie.io	0x0...	0x0...	#@aLC%...	1010101...	1010101...	ToS, Order ID, etc.

The Harpie team has plans to have its database and data security audited later this year.

III: Security Features

Security-first development, always.

The Harpie Team has been dedicated to your security since day one, and it shows in our development choices.

"Not Your Keys, Not Your Coins"

The most significant security feature behind Harpie is that we **never see or store your keys**. We're subscribers to this classic idea, and it shows in our tech and our supported devices.

Our service rests on the `approve` function, which is a keyless authorization process. We support hardware wallets like Ledger and Trezor through MetaMask (and soon will have direct API support as well). Harpie is keyless, and we don't plan on changing.

Minimized Supply Chain and Local APIs

The more APIs and providers a service is connected to, the less secure it is. API jungles at their most harmless cause website slowdowns, and at the most severe cause people to lose their wallets.[8]

Harpie keeps its dependencies to an absolute minimum. We only use the ones that are absolutely necessary. The JavaScript libraries that we do use are publicly shown on our open-source repo, as well as our NuGet packages. Additionally, our database and logins are covered by a single provider: Microsoft Azure.

The few APIs that we do use are all local: this means that we never receive or pull updates from the Internet automatically. It takes a new review process on GitHub to update our APIs

[8] https://snyk.io/blog/a-post-mortem-of-the-malicious-event-stream-backdoor/

each time. This process ensures that malicious code can't infect the Harpie ecosystem, since each pull is open to comment from our open-source community.

We're not the third-party filled mess that you'll see from other blockchain players, and that makes us more secure than the rest.

Salts and Other Entropy

Our encryption algorithm, AES-256-CBC, uses randomly-generated initialization vectors and salts. These are in place to stop rainbow table attacks: attacks on databases that capitalize on commonly-used words. Salts and IVs pad your security question answers with random info; for example, because of salts and IVs, two users answering "Cat, dog, snake" will end up having different encryption keys. Because we use salts and IVs, your encrypted data is protected against rainbow table attacks.

Database Security

Harpie's databases are stored and managed by Microsoft Azure. We make use of the extensive security features provided by the Microsoft Azure service, including virtual network firewalls, TLS and TDE encryption, and regular logging and monitoring of incoming/outgoing network traffic.

Microsoft's Azure SQL Database participates in regular audits, and has been certified against a number of compliance standards.[9] [10]

"What if Harpie goes away?"

A startup going out of business is always a possibility. For our customers, though, this will never be a risk that they need to keep into account.

---

[9] Microsoft Docs: https://docs.microsoft.com/en-us/azure/azure-sql/database/security-overview

[10] Microsoft Audit Reports: https://servicetrust.microsoft.com/ViewPage/MSComplianceGuide

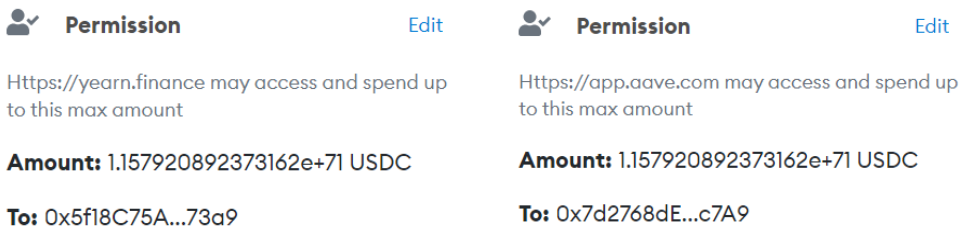
Because Harpie never holds your crypto or your keys, our company dissolving will not result in any loss of crypto or keys. The only thing that a customer will lose is their protection plan with Harpie, since our company will not be around to facilitate transfers.

If our company dissolves, we will make it a priority to destroy the Harpie server data.

Approval Amounts

Recall the `approve` function mentioned earlier.

Here's the industry standard practice with approval:



Yearn Finance and Aave's infinite approval functions.[11] [12]

Seems like a security risk? You're completely correct.[13] A malicious actor that's able to access the recipient account will be able to withdraw every single token you own, because you've approved them for an infinite amount.

Harpie does it differently and allows you to manually input your approval. While this does have some drawbacks in terms of usability, we believe that your security and peace-of-mind is worth protecting above all.

---

[11] Yearn Finance: yearn.finance

[12] Aave: aave.com

[13] CoinMarketCap: https://coinmarketcap.com/alexandria/glossary/infinite-approval

IV: Next Steps

For the people; now and forever.

Harpie was founded in January 2021 and continues to scale up and build as time goes on. At the time of writing, we are helping 20 of our beta testers protect over $6,000 of their tokens.

Right now, Harpie is a three-person project, but we soon plan on seeking investment to expand our team by two or three members. We want Harpie to be a robust interface where any crypto holder can attain peace-of-mind without exposing themselves to security threats. Having a larger team allows us to focus on both building tech and curating a community.

We would love to integrate with wallet providers and exchanges to help provide consumers with token protection, at no cost to the consumers. We also want to do security analyses for common DeFi protocols and publish those as public safety notices.

In one to two years, we plan on releasing Harpie as a public good: an npm repository or executable that can work on non-Internet-connected devices. This will allow developers and security-conscious crypto users to run their own local Harpie nodes if they so wish. If a user would like to use Harpie without the hassle of running a local node, we will provide our service for a small cost on our website. We were inspired by Bitwarden's dedication to providing their own publicly-available password management software to their users, and hope to accomplish the same.

Harpie's future is with the people it serves. People like you. Come join us, and help us build a global network of security-conscious crypto pioneers.

https://discord.gg/3f9QdkXtmb