IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

MAGIC LABS, INC.,

Plaintiff,

v.

HORKOS, INC. d/b/a PRIVY,

Defendant.

C.A. No. 23-967-RGA

JURY TRIAL DEMANDED

FIRST AMENDED COMPLAINT FOR PATENT INFRINGEMENT

Plaintiff Magic Labs, Inc., for its Complaint against Defendant Horkos, Inc. d/b/a/ Privy, alleges as follows:

NATURE OF THE ACTION

1. This is an action for infringement of U.S. Patent No. 11,546,321 (the "'321 Patent", Ex. A) and U.S. Patent No. 11,818,120 (the "'120 Patent", Ex. B) (collectively, the "Asserted Patents").

THE PARTIES

2. Plaintiff Magic Labs, Inc. ("Magic") is a corporation organized in the State of Delaware with its headquarters located at 396 Townsend Street, San Francisco, CA 94107. Magic is the sole owner of the Asserted Patents.

3. Defendant Horkos, Inc. d/b/a/ Privy ("Privy") is a Delaware corporation with its principal place of business located at 228 Park Avenue South, PMB 67932, New York, NY 10003.

JURISDICTION AND VENUE

4. This civil action arises under the patent laws of the United States, 35 U.S.C. §§ 1, *et seq*. This Court has jurisdiction over the subject matter of this action pursuant to 28 U.S.C. §§ 1331 and 1338.

5. This Court has personal jurisdiction over Privy, and venue is proper in this district pursuant to 28 U.S.C. § 1400(b), because Privy is incorporated under the laws of Delaware.

BACKGROUND

6. Magic is a software company founded in 2018 by Fei-yang "Arthur" Jen, Jaemin Jin, and Shang "Sean" Li. Magic enables businesses to adopt blockchain technology with a unique digital asset wallet solution. Magic developed a unique system architecture for generating and maintaining blockchain wallets. With Magic's technology, any business can allow its customers to easily create and manage a secure wallet for digital assets—such as cryptocurrencies or non-fungible tokens ("NFTs"). Those businesses (*i.e.*, Magic's customers) need not have blockchain expertise of their own because Magic's technology handles the complex technological challenges of securely generating and maintaining blockchain wallets, while seamlessly integrating with its customers' existing websites. In January 2023, the United States Patent and Trademark Office ("USPTO") awarded Magic the '321 Patent in recognition of its novel digital asset wallet solution. In November 2023, the USPTO awarded Magic the '120 Patent, which issued from a continuation of the application that led to the '321 Patent, further recognizing its novel innovations in digital asset wallet technology.

7. On information and belief, Privy was founded in 2021 by Henri Stern and Asta Li to leverage blockchain technology for a very different purpose: data storage. It failed to gain traction in the market. Rather than innovating, Privy copied Magic's technology and launched a

competing digital asset wallet solution. Magic brings this case to stop Privy's infringement of its patented technology.

A. Blockchain and Web3

8. Magic and Privy offer "web3" solutions: they help businesses that have historically not relied on blockchain technology to build out their brands in the web3 era. Web3 (or Web 3.0) generally refers to the concept of a new category of Internet services powered by blockchains. Blockchains are decentralized digital ledgers that consist of "blocks" of data in a chain. Blockchain blocks track the ownership and transfer of digital assets on the blockchain network. Each block contains the history of data transactions that have occurred as part of the blockchain. Blockchains utilize public-private key cryptography and software-based consensus mechanisms to ensure security, authenticity, and tamper-resistance for the transactions they facilitate. Due to the cryptography involved, any attempt to change the ledger would fail because it could easily be proven not to be authentic. The decentralized nature of blockchains means that if one computer in the network is compromised, the others contain the ability to reference the entire blockchain and its transaction history. This protects against fraudulent or malicious changes to the ledger.

9. Two of the first web3 applications are cryptocurrencies and NFTs. Cryptocurrencies are digital assets not controlled or distributed by central fiat authorities (such as banks or sovereign treasuries) and are instead generated and transacted via blockchains. Bitcoin, launched in 2009, was the first cryptocurrency. Hundreds of other cryptocurrencies have been released since Bitcoin was launched. More than one in five Americans have been estimated to have traded or owned digital assets.

10. NFTs are another form of digital asset. Each NFT constitutes a unique identifier generated cryptographically. This cryptographic identifier authenticates digital content on a

blockchain. The blockchain's decentralized ledger provides record-keeping, authentication, and proof of ownership of NFTs. NFTs have resulted in the emergence of a new market for one-of-a-kind digital art. Content linked to NFTs can include images, videos, music, or text, among other things.

B. Magic's Fast-Growing Business is Fueled by its Unique Technology Innovation in the Web3 space.

11. Magic has experienced growth in the web3 market. Brands have used its technology to create and secure over 25 million end-user wallets. Its technology is scalable: Magic can create 2,000 wallets per second and support hundreds of millions of users. Brands in the retail, music, fashion, and gaming space have deployed Magic's technology to great success. Seeing Magic's success and the opportunity ahead, top venture capital funds have invested over $80 million dollars into Magic.

12. Magic's software allows consumers who are new to digital assets to create a secure wallet with a few clicks and immediately begin making transactions, and seamlessly integrates that wallet with services and experiences offered by brands.

13. To illustrate, a Magic customer launched an NFT marketplace for digital artwork related to its products. Because Magic's wallet provides a secure and approachable interface for consumers and can be integrated with a brand's systems, the customer was able to create a marketplace for mainstream consumers that did not require users to own cryptocurrency and could be integrated with a peer-to-peer trading platform for consumers to trade their branded NFTs.

C. Privy is Attempting to Build a Business by Knocking Off Magic's Technology.

14. On information and belief, Privy began as a web3 data storage company. Its technology focused on storing data on the blockchain. Privy never succeeded in bringing this technology to market.

15. On information and belief, Privy then pivoted to the digital wallet space. Around that same time, Privy raised $8.3 million in a seed funding round. Privy now markets a digital wallet that directly competes with Magic and copies Magic's patented technology.

16. Privy advertises an "embedded wallet", which is a wallet provided by Privy and allows users "to take wallet-based actions without ever leaving [a brand's] site. Embedded wallets are the *easiest* way to unlock your full product experience for users who don't have, or don't want to connect, their own wallet."[1] This is the same functionality provided by Magic's Universal Wallet, which is a web-based wallet that allows users to login via email or email-based account linking.[2] Privy claims its software, like Magic's, provides "a unified UI [user interface] across a user's various wallets."[3]

17. It is no coincidence that, after failing to find success in data management, Privy looked to Magic's technology in the digital wallet space: Privy's founders are familiar with and have history with Magic. Privy's co-founder Mr. Stern was introduced to a Magic executive by a mutual contact in the fall of 2021. That Magic executive had a call with Mr. Stern later in the fall of 2021. From that call, Mr. Stern learned details about Magic's technology and how Privy could integrate and benefit from that technology.

[1] https://docs.privy.io/guide/frontend/embedded/overview.

[2] https://magic.link/docs/universal/resources/faqs.

[3] https://www.privy.io/blog/launching-privy.

18. In February 2022, Magic's co-founder Mr. Jen attended an event with Mr. Stern. At that event, Mr. Jen and Mr. Stern spoke in detail about Privy's technology. Mr. Stern acknowledged that he had been aware of Magic's technology and thought it was great—and, in fact, he had modeled Privy's technology on Magic's technology.

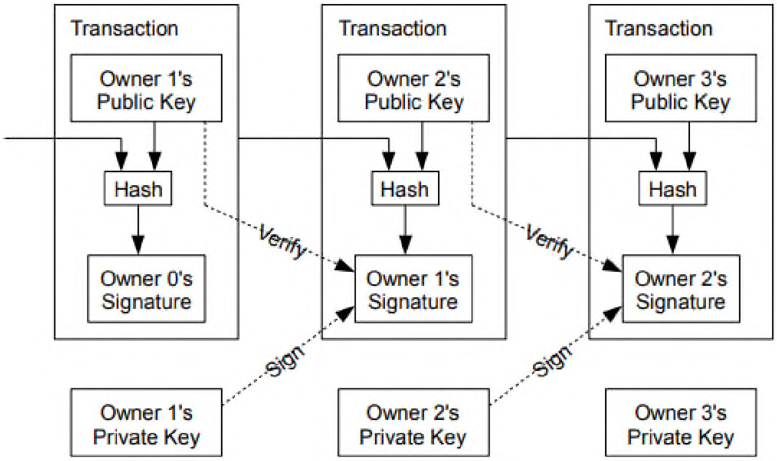
19. On information and belief, in April 2022, Privy closed its $8.3 million seed funding round.

MAGIC'S PATENTED TECHNOLOGY

20. The Asserted Patents relate to novel methods and systems for authenticating an end user—for example, for generating and storing private keys in a digital wallet in a non-custodial manner. The '321 Patent issued on January 3, 2023 from an application filed with the USPTO on September 24, 2020, which claims priority to a provisional application filed on September 24, 2019. The '120 Patent issued on November 14, 2023 from an application filed with the USPTO on December 30, 2022. The '120 Patent is in the same patent family as the '321 Patent and claims priority to the same provisional application.

21. Many web3 applications are built on the principles set forth in a white paper published by Bitcoin in 2008. Specifically, the white paper described a system wherein each transaction is recorded using a public key and a private key from each owner in the chain of transactions:[4]

[4] https://bitcoin.org/bitcoin.pdf.



22. The public key acts as a wallet address for the owner to allow them to receive inbound transactions—like a bank routing number. The private key is held by the owner and is used to sign transactions and prove ownership of funds—like a bank account number. Each owner's public key is derived from their private key via a cryptographic algorithm that cannot be reversed to reveal the private key.

23. Public-private key systems existed long before the Bitcoin white paper was published. But cryptocurrency and other blockchain-based digital assets changed the role of those keys: whereas a private key previously provided access to an asset, a private key itself is essentially the asset in cryptocurrency because ownership is determined by the private key. In other words, whoever holds the private key owns the digital asset—whether cryptocurrency, an NFT, or some other asset associated with that key.

24. The Bitcoin white paper did not provide a solution for managing private keys. By 2015, cryptocurrency had broken into the mainstream, but managing, controlling, and using cryptographic keys remained complex tasks, and no clear solution had been proposed. Those complex tasks involved, for example, generating new keys, changing and deleting existing keys, identifying the end user associated with a key, authenticating the end user as the person he or she

claims to be, and confirming the end user is allowed access to the key in question—all while maintaining the highest level of security.

25. Prior to Magic's invention, there were essentially four options for performing these key generation and management tasks: (1) using a standalone, physical computing device known as a hardware security module ("HSM"); (2) using a third-party HSM accessible through a cloud environment; (3) using a third-party server; or (4) using a browser, browser extension, or mobile application. Each option had serious drawbacks.

26. HSMs are computing devices specifically designed to generate and store cryptographic keys. While a personal HSM would eliminate the need to identify and authenticate multiple end users, HSMs are not consumer-grade products and cost thousands if not tens-of-thousands of dollars. This is not a realistic option for most end users. Moreover, even if you were able to afford such a solution, a HSM is a large physical device. This presents a variety of logistical and security concerns: in order to access your private keys (and thus conduct digital asset transactions), you would need to have your HSM physically with you. But, at the same time, carrying private keys for digital assets on your physical person is tantamount to carrying all the money in your bank account on your physical person; few end users would feel comfortable carrying their HSM with them in their day-to-day lives (even if they were strong enough to lift one).

27. Third-party HSMs allow multiple end users to store and encrypt private keys on a HSM operated in a secure cloud environment and access those keys through the third-party's service. But this places the onus of generating keys and coordinating with the third-party provider on the end user, including to ensure that you and only you have access and control over your keys in the secure cloud environment. And as a practical matter, such third-party HSMs are

typically meant to serve enterprises and are complicated and expensive to set up—generally requiring direct application programming interface ("API") access. Thus, few end users would consider this a real option. For consumer-grade HSMs, they still are difficult for end-users to manage and do not enable seamless blockchain application access.

28. Private keys also could have been generated and stored on a centralized server. Because this solution does not require unique end user hardware, it is lower cost and can be used with direct-to-consumer services. But generating and storing private keys on a centralized server goes against the core tenets of web3, which is built on decentralization secured by blockchains. Indeed, Bitcoin was meant to provide a means for conducting transactions without relying on "a trusted central authority."[5] Allowing one entity—the entity that controls the server—to hold the end user's private keys makes that entity a "trusted central authority" just like a bank or mint. Not only is this solution contrary to the ethos of web3, but it also raises security concerns: for example, the server entity's personnel will have access to others' private keys and may abuse that information, and the centralized server is likely to become a target for hackers.

29. Alternatively, private keys could have been generated and stored in a browser, browser extension, or mobile app. Although this approach did not require specialized hardware or rely on a central authority, it was plagued with security vulnerabilities because it exposed the keys to attacks via exploits, code injections, side channel attacks, in-browser memory compromises, or other vulnerability vectors.

30. A problem common to all four of these solutions—holding a large HSM, using a third-party cloud HSM, generating a key on a server, or generating a key in the browser—is that there is a direct computer networking or memory connection link between the software library

[5] https://bitcoin.org/bitcoin.pdf at 2.

and process that generates the key and the application that is using the key. Consequently, the key in these solutions is passed back and forth and exposed to malicious insiders or external attackers. The key will at some point exist on the centralized system providing services to the user. If it is compromised in transit (e.g., through a network interception), in storage, or in memory, the underlying digital assets are at risk or underlying computer systems can be maliciously taken over. For these reasons, it is not possible to provide a trustless authentication solution with any of the four prior approaches.

31. Magic's claimed inventions address these existing problems with an elegant solution: as summarized in the Asserted Patents' shared specification,[6] the patented technology "is non-custodial, wherein a public-private key pair, which represents user identity, is created on a client machine and then directly encrypted by a third-party platform without relying on one centralized computing system." '321 Patent at 1:51–55.

32. In other words, Magic invented a novel system architecture that decentralizes generating and storing cryptographic keys to provide end users with the security and control benefits of a hardware solution with the convenience of a software solution. Prior to Magic's invention, industry solutions either prioritized security over convenience by implementing a hardware solution (a personal HSM or third-party HSM) that placed the substantial burden of managing key generation and storage on the end user; or prioritized convenience over security by implementing a software solution (centralized server or browser, browser extension, or mobile app) that took on the management burden but exposed the end user's keys to bad actors.

[6] The '321 Patent and '120 Patent share substantially the same specification. For ease of reference, Magic cites to the '321 Patent specification herein.

33. Magic solved this problem by inventing a new system architecture that inverted the conventional industry architectures. Instead of having one entity (either the end user or a software service provider) manage key generation and storage, the various underlying tasks could be divided and delegated between the end user, a software service provider, and a third-party key storage provider. The software service provider acts as a *non-custodial* intermediary between the end user and third-party key storage provider, providing the infrastructure the user needs to securely generate keys and coordinate with a third-party key storage provider while allowing the user to retain complete control of his or her keys.

34. Figure 2A of the Asserted Patents illustrates one embodiment of this novel architecture.

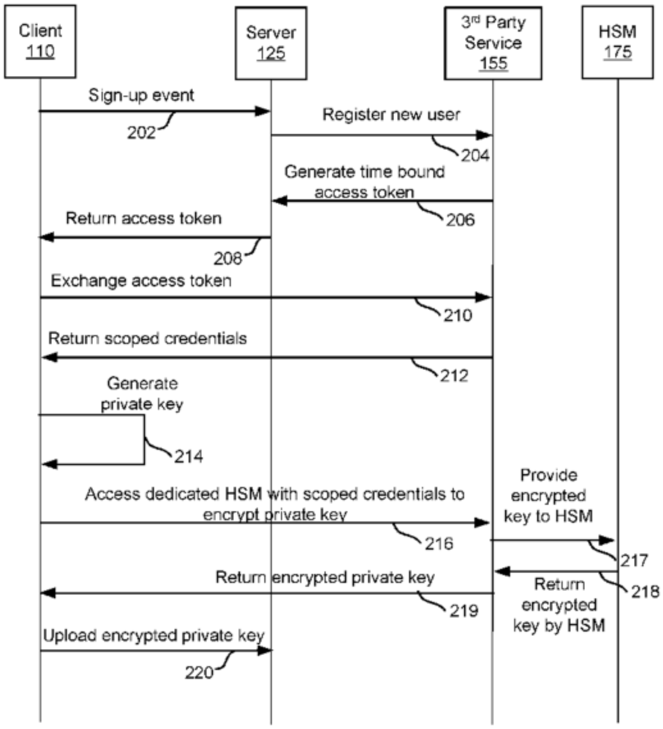


FIGURE 2A

35. In the embodiment shown in Figure 2A, the client can sign up (202) through the authentication server (125), for example, by "sending an email to an email account associated

with the user that initiated the sign-up event." '321 Patent at 4:34–45. After the user is authenticated by the server (125), the user is provided an "access token" that allows him or her to communicate directly with the third-party service (155) that is linked to the HSM (175). *Id.* at 4:46–55. When the third-party service receives the access token, it generates a master key and scoped credentials, and provides the scoped credentials to the user (212). *Id.* at 4:56–61. As the specification explains:

> By providing the scoped credentials to the user, the user is enabled to work with the third-party service to access their master keys stored on the third party service, for example for encryption and decryption. The server is bypassed in this step, and cannot forge or intercept the scoped credentials. In some instances, the access token and scoped credentials can be created dynamically by the third-party service, with audit logs and with a time-to-live (TTL) enabled.

Id. at 4:61–5:2.

36. In other words, the system is designed to allow the third-party service to provide credentials for accessing the master key used to encrypt and decrypt the user's keys directly to the user. This is done without exposing those credentials to the authentication server or the master key to any technology infrastructure outside the third-party service. As a result, the user never cedes control of its credentials or keys to the authentication server—despite the fact that the user signed up through the authentication server and received the "access token" for communicating with the third-party service from the authentication server. *See id.*

37. In this system, after receiving the scoped credentials, the user generates a public-private key pair at the user's device (214) using those credentials. Notably, the key pair is generated by the user's device (client) without exposing the keys to the authentication server (125). The invention accomplishes this by generating the keys in a secure environment within a browser on the user's device—for example, "within a JavaScript iframe implemented within a

network browser at the client," which would not be accessible to any application associated with the platform operator on the server side. *Id.* at 5:3–13. The iframe solution also solves the technical challenge of generating keys in a browser and exposing those keys to in-browser memory and other side-channel attacks.

38. By isolating the client-side key-generation function from the rest of the content within the browser—e.g., through the use of an iframe, the invention provides a technological solution (via sandboxing) to a technological problem (i.e., potential security risks posed by generating keys within a portion of the browser that could be manipulated/accessed by malicious code). This technological solution of securely generating the key pair client side likewise means that the user does not need to trust the platform operator to securely use the service. Because, from a technological perspective, the iframe operates like a browser within a browser in this context, it means the user does not even need to trust their own web browser or computer outside of the iframe.

39. The user then sends the public-private key pair and credentials directly to the third-party service (155), bypassing the authentication server (125). The third-party service then encrypts the user's keys with its master key and provides the user's encrypted keys to the HSM (217). It also returns the user's encrypted keys directly to the user (219), and the user provides the encrypted keys to the authentication server (220). Because the user's keys can only be decrypted with the third-party service—which generated and stores the master key used to encrypt and decrypt the user's keys, and never shares the master key with any other entity—the authentication server never has access to the user's unencrypted public-private key pair. *Id.* at 5:3–31.

FIRST CAUSE OF ACTION

(INFRINGEMENT OF U.S. PATENT NO. 11,546,321)

40. Magic incorporates by reference and re-states paragraphs 1 through 39.

41. Magic is the owner of all right, title, and interest in and to the ’321 Patent.

42. Privy directly infringes the ’321 Patent at least through making, using, testing, offering to sell, selling, and/or importing into the United States its wallet products and related services, including the embedded wallet product (together, “the Accused System”) and its prototypes and/or related functionalities.

43. Privy offers the Accused System for sale on its website.[7] On information and belief, Privy develops the Accused System in the United States and offers to sell, sells, tests, uses, and/or imports into the United States the Accused System. Privy operates and is headquartered at 228 Park Avenue South, PMB 67932, New York NY 10003, where, on information and belief, it develops the Accused System and offers to sell, sells, tests, and/or uses the same.

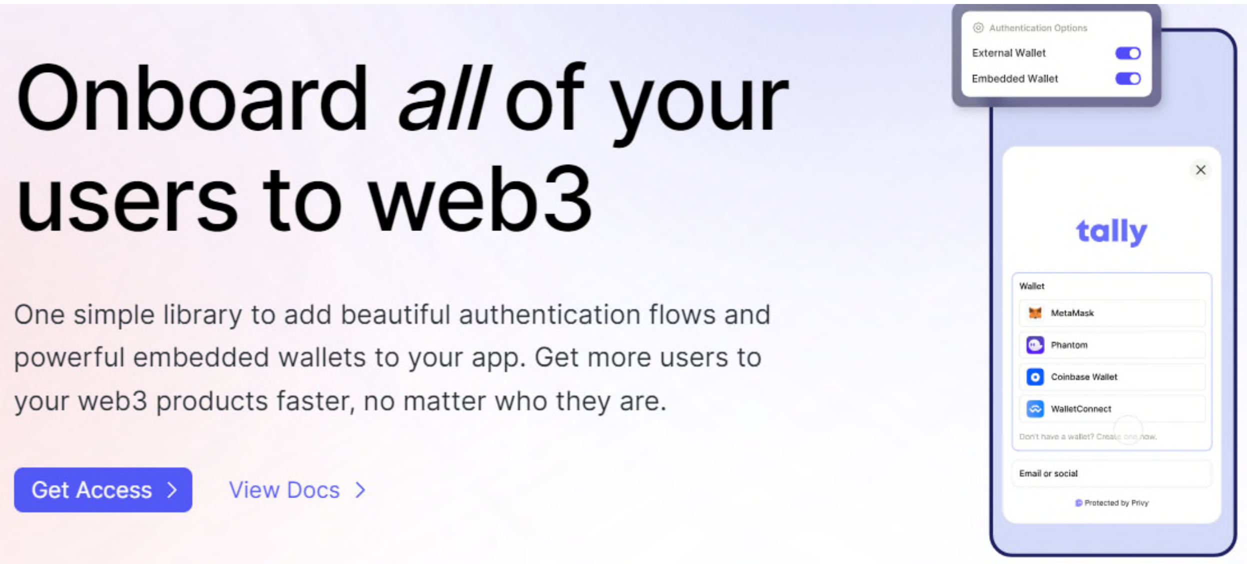
44. The Accused System infringes the ’321 Patent. For example, the Accused System meets each limitation of at least claim 11 of the ’321 Patent.

45. To the extent the preamble is limiting, the Accused System comprises “[a] non-transitory computer readable storage medium having embodied thereon a program, the program being executable by a processor to perform a method to setup a wallet for a decentralized application by performing a non-custodial authentication method for a client.” ’321 Patent, claim 11. The Accused System includes a program, code, and/or a software development kit (“SDK”) that is obtained and accessed via a computer (e.g. a personal computer or “PC”). As

[7] https://www.privy.io/pricing

program, code, and/or an SDK, the Accused System is designed to be stored on a non-transitory computer readable storage medium having embodied thereon a program, the program being executable by a processor. The Accused System also employs servers and other backend computers to implement the claimed functionality.

46. The Accused System is a software application that allows users to setup a decentralized web3 wallet and perform authentication flows:



Onboard *all* of your users to web3

One simple library to add beautiful authentication flows and powerful embedded wallets to your app. Get more users to your web3 products faster, no matter who they are.

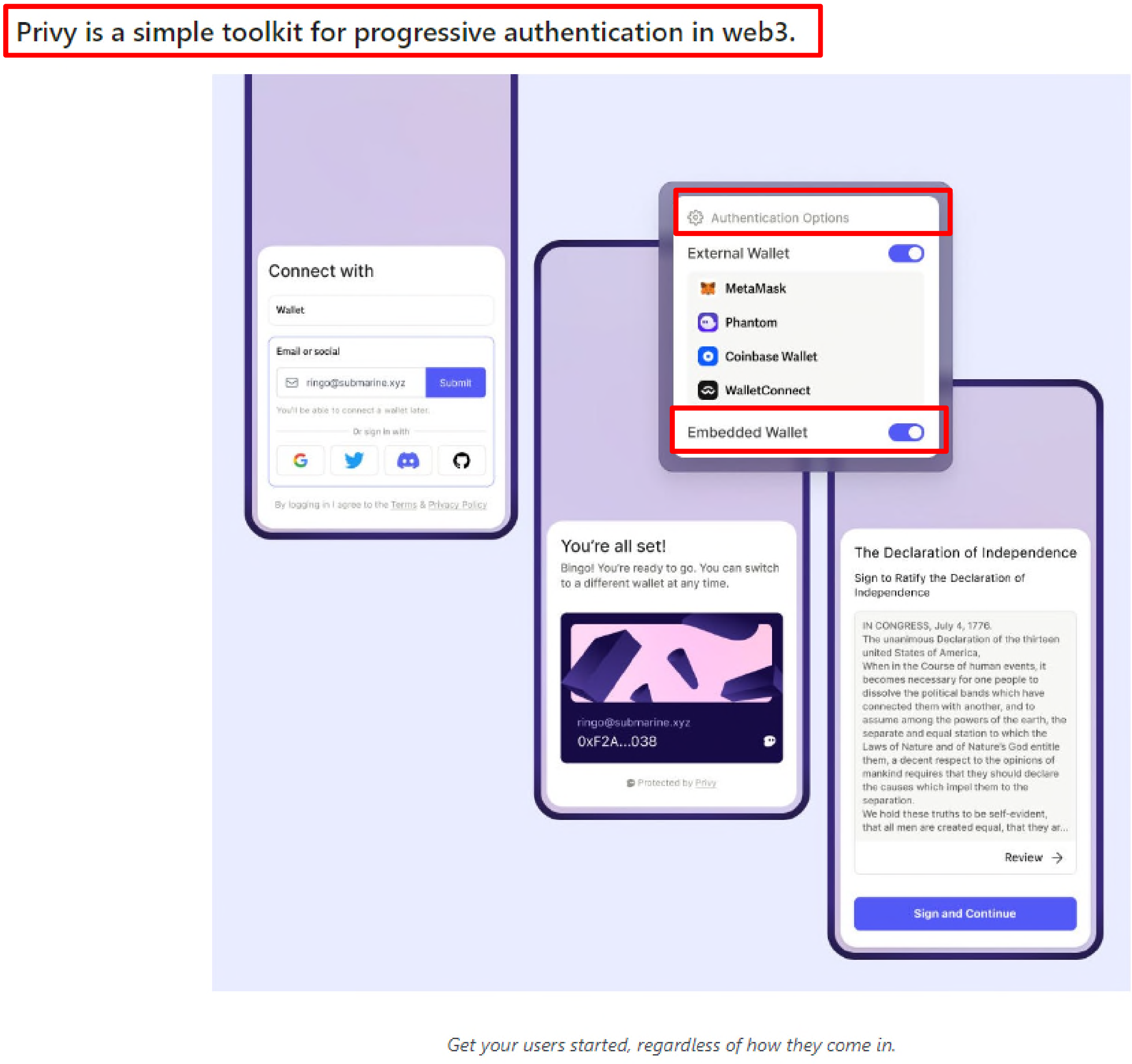
Authentication Options

External Wallet ●

Embedded Wallet ●

https://www.privy.io/.

47. As shown below, the authentication flows are non-custodial because Privy does not have access to the protected user keys or data, as the keys and data are encrypted by the user's client and the third-party HSM service that Privy uses:

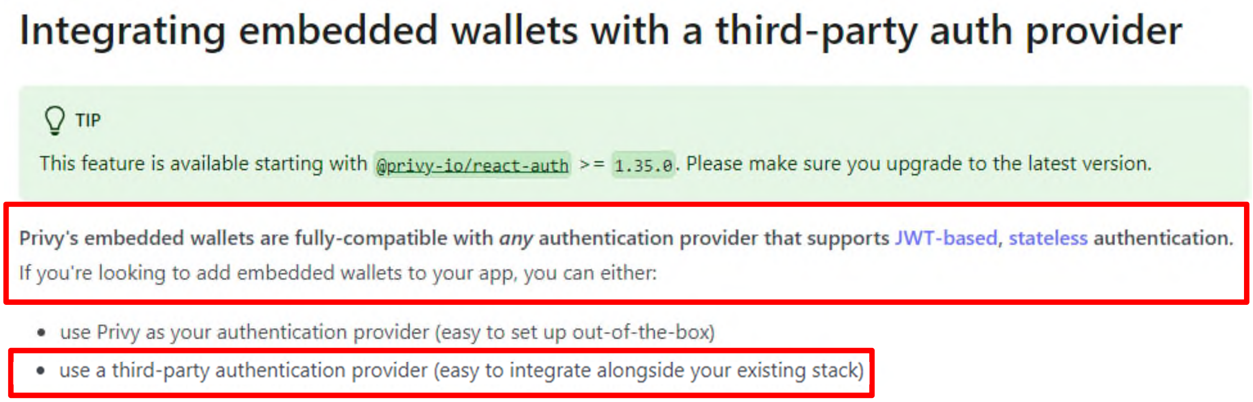


Privy is a simple toolkit for progressive authentication in web3.

IN CONGRESS, July 4, 1776.
The unanimous Declaration of the thirteen united States of America,
When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.
We hold these truths to be self-evident, that all men are created equal, that they ar...

Get your users started, regardless of how they come in.

https://docs.privy.io/#privy-is-a-simple-toolkit-for-progressive-authentication-in-web3.



Integrating embedded wallets with a third-party auth provider

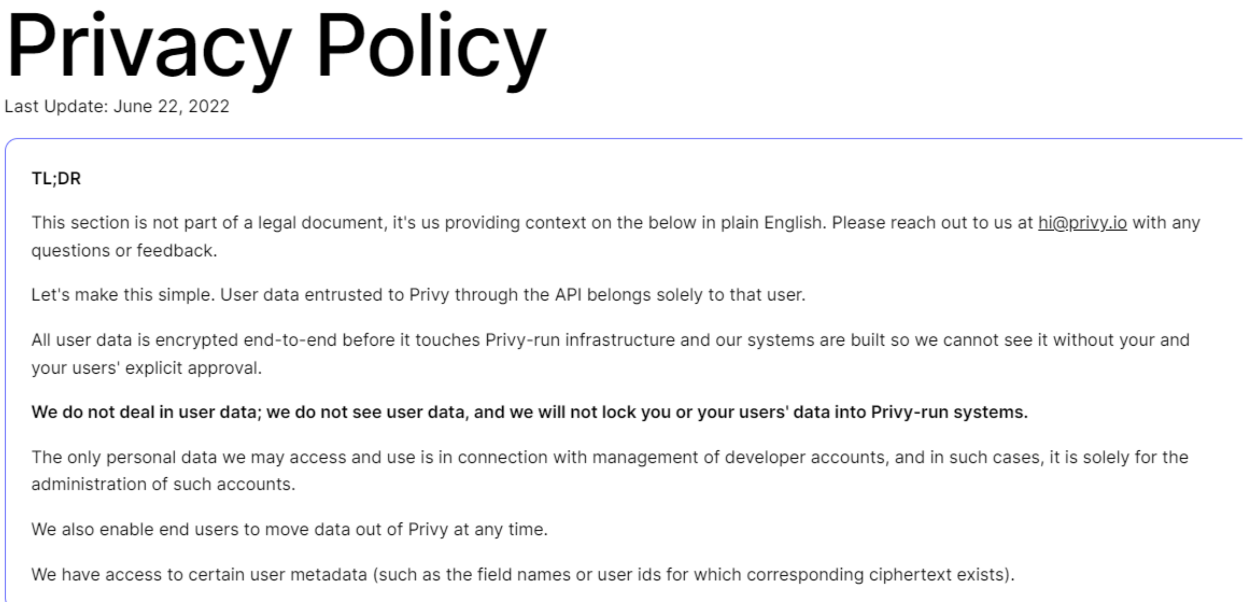
TIP

This feature is available starting with `@privy-io/react-auth` >= `1.35.0`. Please make sure you upgrade to the latest version.

Privy's embedded wallets are fully-compatible with *any* authentication provider that supports JWT-based, stateless authentication. If you're looking to add embedded wallets to your app, you can either:

- use Privy as your authentication provider (easy to set up out-of-the-box)
- use a third-party authentication provider (easy to integrate alongside your existing stack)

https://docs.privy.io/guide/guides/third-party-auth.



Privacy Policy

Last Update: June 22, 2022

TL;DR

This section is not part of a legal document, it's us providing context on the below in plain English. Please reach out to us at hi@privy.io with any questions or feedback.

Let's make this simple. User data entrusted to Privy through the API belongs solely to that user.

All user data is encrypted end-to-end before it touches Privy-run infrastructure and our systems are built so we cannot see it without your and your users' explicit approval.

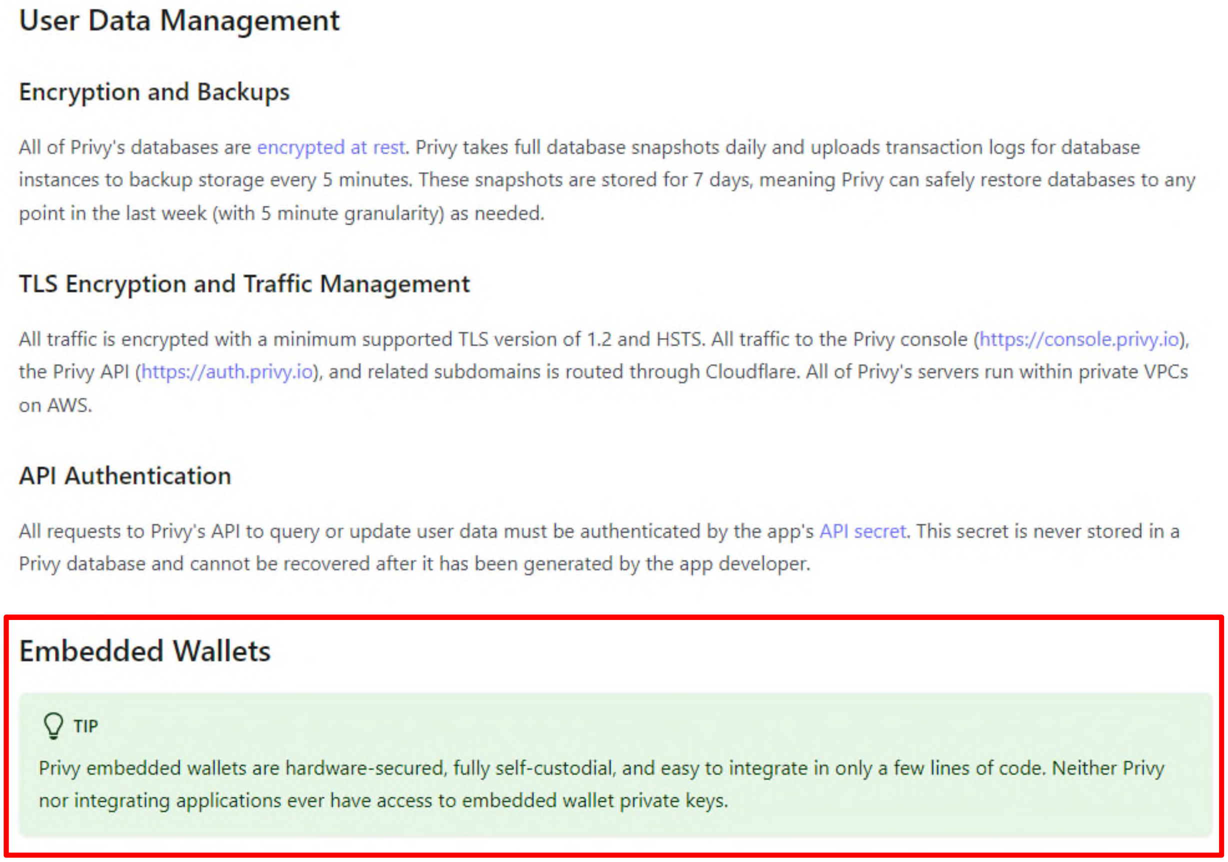
We do not deal in user data; we do not see user data, and we will not lock you or your users' data into Privy-run systems.

The only personal data we may access and use is in connection with management of developer accounts, and in such cases, it is solely for the administration of such accounts.

We also enable end users to move data out of Privy at any time.

We have access to certain user metadata (such as the field names or user ids for which corresponding ciphertext exists).

https://www.privy.io/privacy-policy.



User Data Management

Encryption and Backups

All of Privy's databases are encrypted at rest. Privy takes full database snapshots daily and uploads transaction logs for database instances to backup storage every 5 minutes. These snapshots are stored for 7 days, meaning Privy can safely restore databases to any point in the last week (with 5 minute granularity) as needed.

TLS Encryption and Traffic Management

All traffic is encrypted with a minimum supported TLS version of 1.2 and HSTS. All traffic to the Privy console (https://console.privy.io), the Privy API (https://auth.privy.io), and related subdomains is routed through Cloudflare. All of Privy's servers run within private VPCs on AWS.

API Authentication

All requests to Privy's API to query or update user data must be authenticated by the app's API secret. This secret is never stored in a Privy database and cannot be recovered after it has been generated by the app developer.

Embedded Wallets

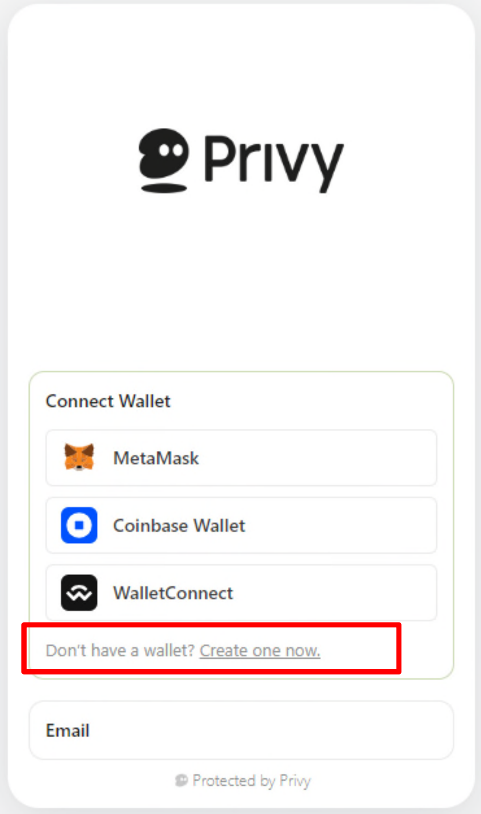
TIP

Privy embedded wallets are hardware-secured, fully self-custodial, and easy to integrate in only a few lines of code. Neither Privy nor integrating applications ever have access to embedded wallet private keys.

https://docs.privy.io/guide/security#user-data-management.[8]

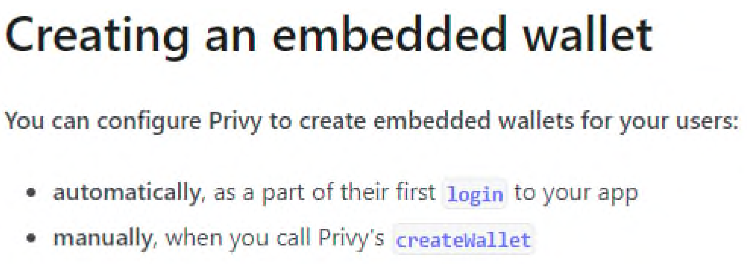
48. The Accused System "send[s], over a network by the client to an authentication system, a sign-up request for a user account associated with the decentralized application." '321 Patent, claim 11.

49. The Accused System prompts the user to sign up by creating a wallet for a user account associated with a decentralized, wallet application once the client has opened the Privy interface:



https://demo.privy.io/.

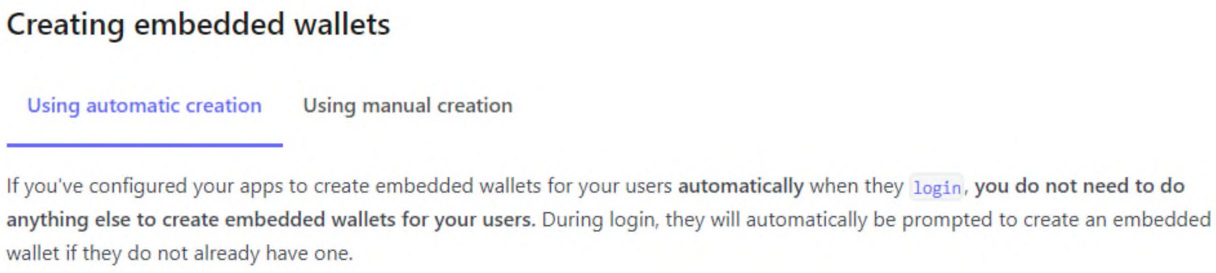
[8] Some of the images from Privy's website reproduced herein were captured prior to the filing of the initial Complaint in this action on September 5, 2023. It appears that Privy has since made superficial changes to portions of its website—for example, by removing the phrase "hardware-secured" from the description of "Embedded Wallets." On information and belief, Privy's modifications to its website were made in an attempt to conceal its infringement and Privy continues to use the functionality described in the pre-filing images.



Creating an embedded wallet

You can configure Privy to create embedded wallets for your users:

- **automatically**, as a part of their first `login` to your app
- **manually**, when you call Privy's `createWallet`

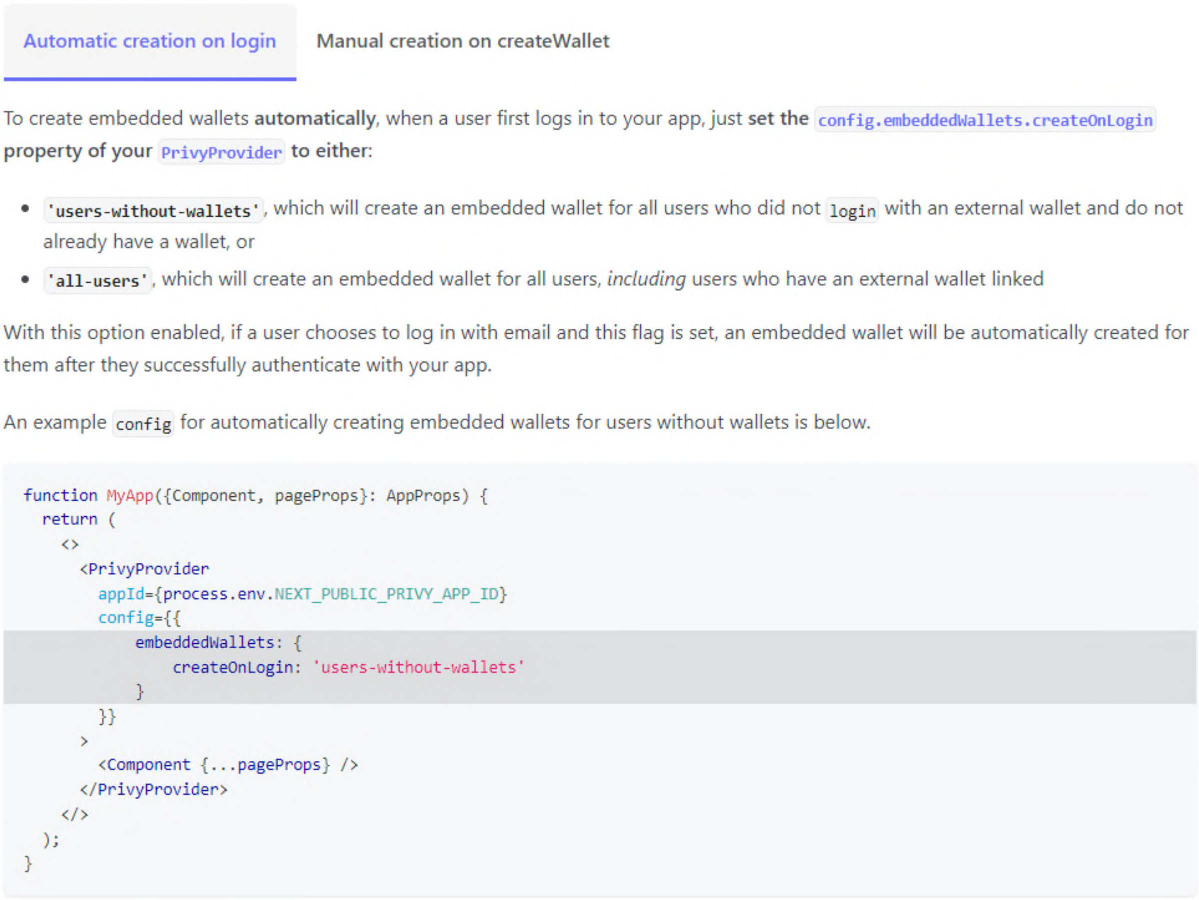


Creating embedded wallets

If you've configured your apps to create embedded wallets for your users **automatically** when they `login`, **you do not need to do anything else to create embedded wallets for your users.** During login, they will automatically be prompted to create an embedded wallet if they do not already have one.

https://docs.privy.io/guide/frontend/embedded/creation/login#creating-embedded-wallets.

50. To sign up for the decentralized application, the client sends the sign-up request to an authentication system:



To create embedded wallets **automatically**, when a user first logs in to your app, just **set the** `config.embeddedWallets.createOnLogin` **property of your** `PrivyProvider` **to either:**

- `'users-without-wallets'`, which will create an embedded wallet for all users who did not `login` with an external wallet and do not already have a wallet, or
- `'all-users'`, which will create an embedded wallet for all users, *including* users who have an external wallet linked

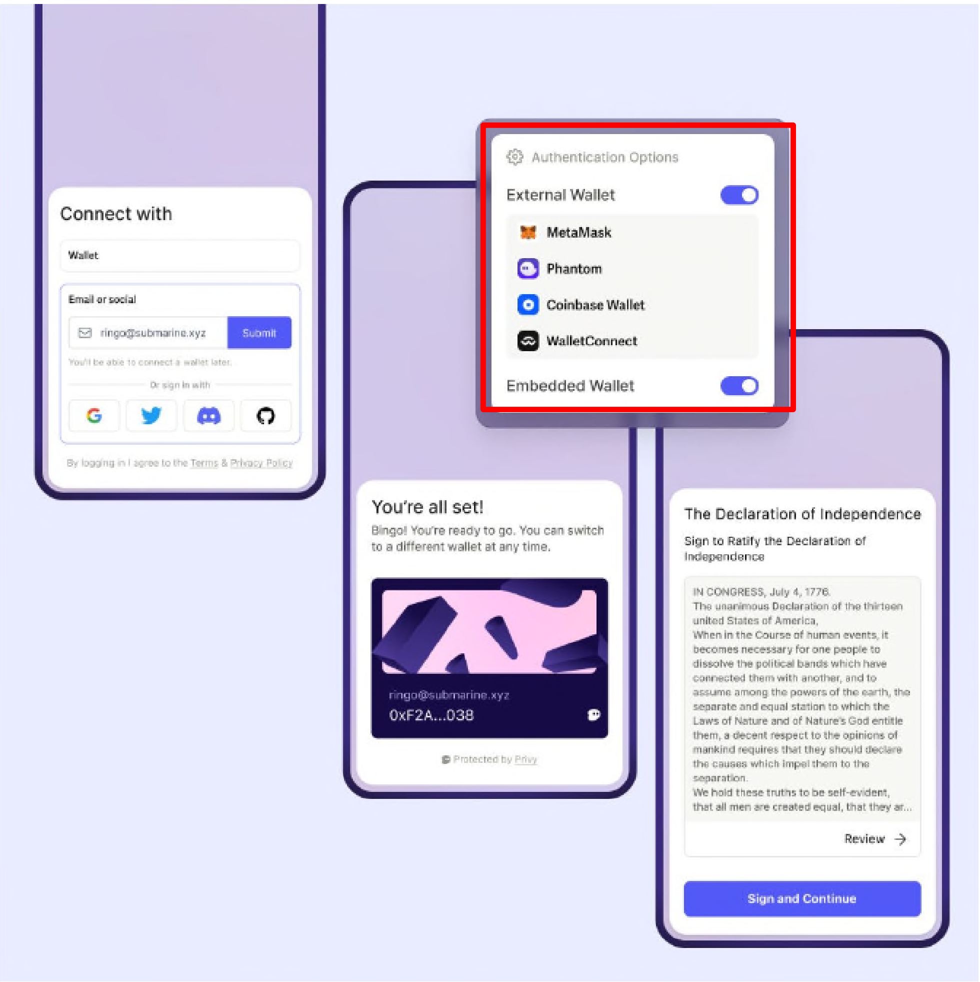
With this option enabled, if a user chooses to log in with email and this flag is set, an embedded wallet will be automatically created for them after they successfully authenticate with your app.

An example `config` for automatically creating embedded wallets for users without wallets is below.

```
function MyApp({Component, pageProps}: AppProps) {
  return (
    <>
      <PrivyProvider
        appId={process.env.NEXT_PUBLIC_PRIVY_APP_ID}
        config={{
            embeddedWallets: {
                createOnLogin: 'users-without-wallets'
            }
        }}
      >
        <Component {...pageProps} />
      </PrivyProvider>
    </>
  );
}
```

https://docs.privy.io/guide/frontend/embedded/configuration#when-the-wallet-is-created.

Privy is a simple toolkit for progressive authentication in web3.



Connect with

Wallet

Email or social

ringo@submarine.xyz

You'll be able to connect a wallet later.

Or sign in with

By logging in I agree to the Terms & Privacy Policy

Authentication Options

External Wallet: on

MetaMask

Phantom

Coinbase Wallet

WalletConnect

Embedded Wallet: on

You're all set!

Bingo! You're ready to go. You can switch to a different wallet at any time.

ringo@submarine.xyz

0xF2A...038

Protected by Privy

The Declaration of Independence

Sign to Ratify the Declaration of Independence

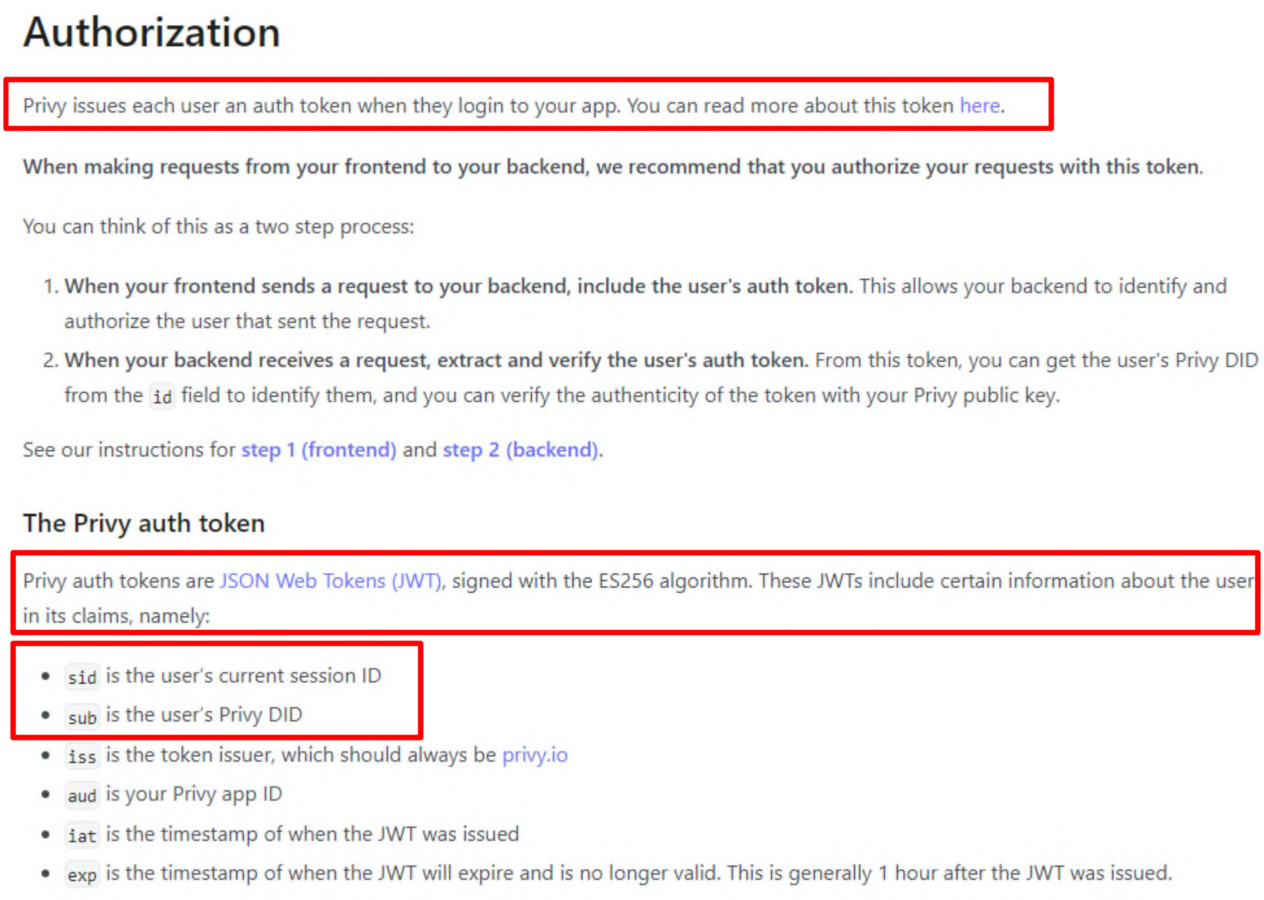
IN CONGRESS, July 4, 1776.
The unanimous Declaration of the thirteen united States of America,
When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.
We hold these truths to be self-evident, that all men are created equal, that they ar...

Get your users started, regardless of how they come in.

https://docs.privy.io/#privy-is-a-simple-toolkit-for-progressive-authentication-in-web3.

51. The Accused System "receiv[es] over the network at the client from the authentication system, an access token that corresponds to the sign-up request, for use at a third party key storage system." '321 Patent, claim 11.

52. The Accused System provides for the client network to receive an access token corresponding to the sign-up request:



Authorization

Privy issues each user an auth token when they login to your app. You can read more about this token here.

When making requests from your frontend to your backend, we recommend that you authorize your requests with this token.

You can think of this as a two step process:

1. **When your frontend sends a request to your backend, include the user's auth token.** This allows your backend to identify and authorize the user that sent the request.
2. **When your backend receives a request, extract and verify the user's auth token.** From this token, you can get the user's Privy DID from the `id` field to identify them, and you can verify the authenticity of the token with your Privy public key.

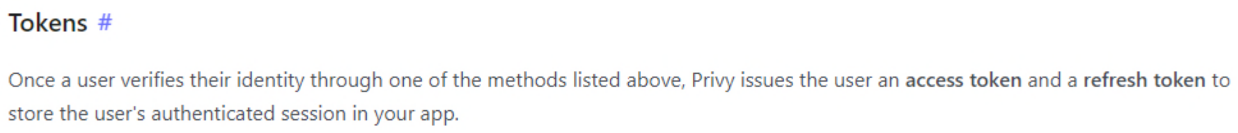
See our instructions for **step 1 (frontend)** and **step 2 (backend)**.

The Privy auth token

Privy auth tokens are JSON Web Tokens (JWT), signed with the ES256 algorithm. These JWTs include certain information about the user in its claims, namely:

- `sid` is the user's current session ID
- `sub` is the user's Privy DID
- `iss` is the token issuer, which should always be privy.io
- `aud` is your Privy app ID
- `iat` is the timestamp of when the JWT was issued
- `exp` is the timestamp of when the JWT will expire and is no longer valid. This is generally 1 hour after the JWT was issued.

https://docs.privy.io/guide/authorization/overview.

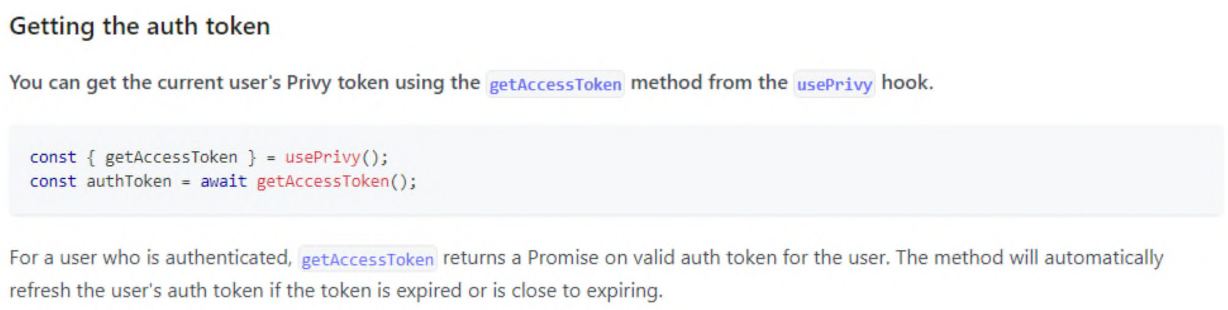


Tokens

Once a user verifies their identity through one of the methods listed above, Privy issues the user an **access token** and a **refresh token** to store the user's authenticated session in your app.

https://docs.privy.io/guide/security#tokens.

53. The Accused System further provides for this access token to be used at a third-party key storage system:



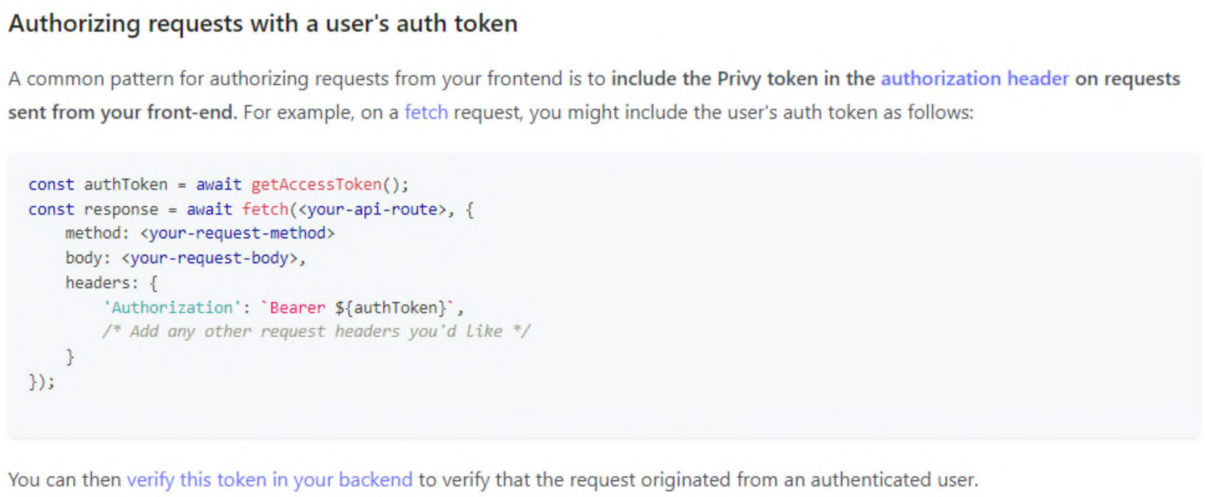
Getting the auth token

You can get the current user's Privy token using the `getAccessToken` method from the `usePrivy` hook.

```
const { getAccessToken } = usePrivy();
const authToken = await getAccessToken();
```

For a user who is authenticated, `getAccessToken` returns a Promise on valid auth token for the user. The method will automatically refresh the user's auth token if the token is expired or is close to expiring.

https://docs.privy.io/guide/frontend/authorization#getting-the-auth-token.



Authorizing requests with a user's auth token

A common pattern for authorizing requests from your frontend is to **include the Privy token in the authorization header on requests sent from your front-end.** For example, on a fetch request, you might include the user's auth token as follows:

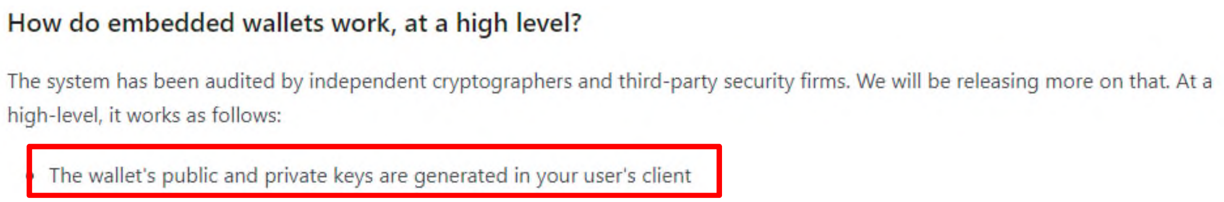
```
const authToken = await getAccessToken();
const response = await fetch(<your-api-route>, {
    method: <your-request-method>
    body: <your-request-body>,
    headers: {
        'Authorization': `Bearer ${authToken}`,
        /* Add any other request headers you'd like */
    }
});
```

You can then verify this token in your backend to verify that the request originated from an authenticated user.

https://docs.privy.io/guide/frontend/authorization#authorizing-requests-with-a-users-auth-token.

54. The Accused System "generat[es] a key by the client." '321 Patent, claim 11.

55. The Accused System provides for a key to be generated by the client:

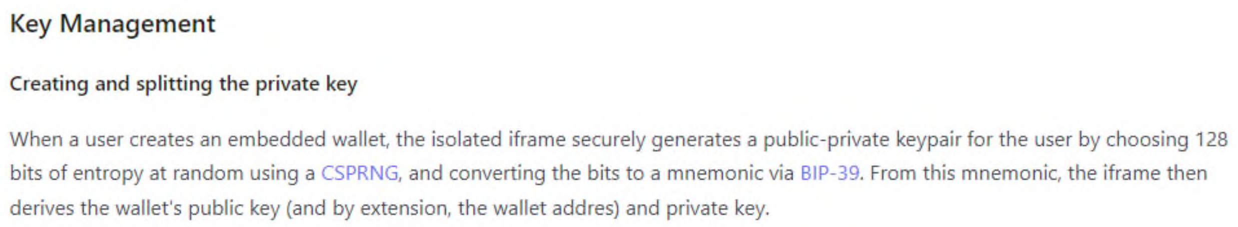


How do embedded wallets work, at a high level?

The system has been audited by independent cryptographers and third-party security firms. We will be releasing more on that. At a high-level, it works as follows:

- The wallet's public and private keys are generated in your user's client

https://docs.privy.io/guide/frontend/embedded/faq#how-do-embedded-wallets-work-at-a-high-level.



Key Management

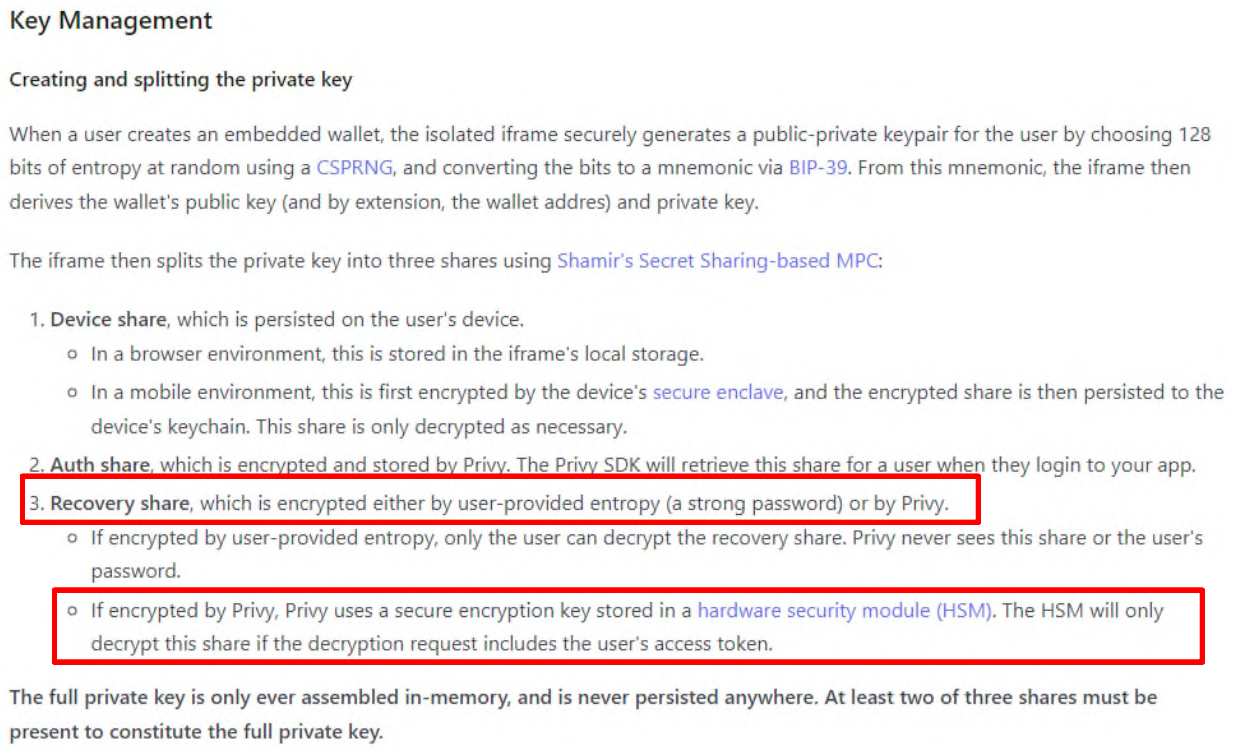
Creating and splitting the private key

When a user creates an embedded wallet, the isolated iframe securely generates a public-private keypair for the user by choosing 128 bits of entropy at random using a CSPRNG, and converting the bits to a mnemonic via BIP-39. From this mnemonic, the iframe then derives the wallet's public key (and by extension, the wallet addres) and private key.

https://docs.privy.io/guide/security#key-management.

56. The Accused System "send[s] over the network from the client to the third party key storage system and bypassing the authentication system, one or more messages that include the access token, the key, and a request to encrypt the key." '321 Patent, claim 11.

57. The Accused System allows for the client network to send a request to encrypt the key, which is called the "Recovery Share," to a third-party HSM. The Accused System also provides the client with the ability to retrieve the key and decrypt it from the HSM



Key Management

Creating and splitting the private key

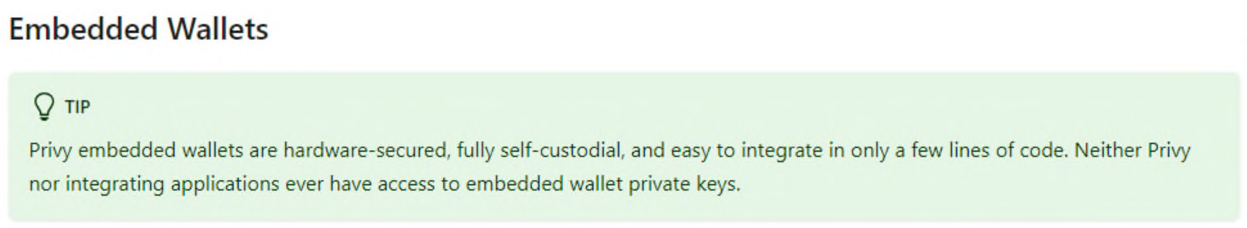
When a user creates an embedded wallet, the isolated iframe securely generates a public-private keypair for the user by choosing 128 bits of entropy at random using a CSPRNG, and converting the bits to a mnemonic via BIP-39. From this mnemonic, the iframe then derives the wallet's public key (and by extension, the wallet addres) and private key.

The iframe then splits the private key into three shares using Shamir's Secret Sharing-based MPC:

1. **Device share**, which is persisted on the user's device.
   - In a browser environment, this is stored in the iframe's local storage.
   - In a mobile environment, this is first encrypted by the device's secure enclave, and the encrypted share is then persisted to the device's keychain. This share is only decrypted as necessary.
2. **Auth share**, which is encrypted and stored by Privy. The Privy SDK will retrieve this share for a user when they login to your app.
3. **Recovery share**, which is encrypted either by user-provided entropy (a strong password) or by Privy.
   - If encrypted by user-provided entropy, only the user can decrypt the recovery share. Privy never sees this share or the user's password.
   - If encrypted by Privy, Privy uses a secure encryption key stored in a hardware security module (HSM). The HSM will only decrypt this share if the decryption request includes the user's access token.

The full private key is only ever assembled in-memory, and is never persisted anywhere. At least two of three shares must be present to constitute the full private key.

https://docs.privy.io/guide/security#key-management.



Embedded Wallets

TIP

Privy embedded wallets are hardware-secured, fully self-custodial, and easy to integrate in only a few lines of code. Neither Privy nor integrating applications ever have access to embedded wallet private keys.

https://docs.privy.io/guide/security#embedded-wallets.

58. Privy also has and continues to induce and/or contribute to the infringement of the'321 Patent by inducing its customers to directly infringe the '321 Patent (for example, when customers put into use the Accused System as described in paragraphs 40 to 53 above), and by contributing to such direct infringement.

59. Privy has known of the '321 patent and its infringement at least September 1, 2023, when Magic sent Privy a letter notifying it of the '321 patent and its infringement.

60. As illustrated in paragraphs 40 to 53 above, Privy actively encourages its users to infringe claims of the '321 Patent by providing the Accused System, advertising how the Accused System can be used in an infringing manner on its website and reference documents, and advertising and advising its users how to incorporate the Accused System into their applications in a manner that infringes the '321 patent.

61. On information and belief, the Accused System has no substantial non-infringing uses because the Accused System is specifically designed to infringe. Privy advertises on the home page of its website that one of the core features of the Accused System is that it allows for "beautiful authentication flows" and the ability to "bring[] web2-caliber UX—like sign in with email and social—to web3 products."[9] In other words, the Accused System allows users to more easily and securely sign up for a decentralized wallet. As explained in paragraphs 40 to 53

[9] https://www.privy.io/

above, this core functionality of the Accused System is enabled by architecture that meets each element of at least claim 11 of the ’321 Patent.

62. On information and belief, Privy will continue to infringe, induce infringement of, and/or contribute to infringement of the ’321 Patent unless enjoined by this Court.

63. As a result of Privy’s infringement of the ’321 Patent, Magic has suffered and will continue to suffer damages in an amount to be proven at trial.

64. Magic is entitled to recover damages for pre-suit infringement because it has complied with 35 U.S.C. § 287, including by providing Privy with actual notice of Magic’s claim that Privy infringes the ’321 Patent. As a result of Privy’s infringement of the Asserted Patents, Magic has suffered and will continue to suffer irreparable harm unless Privy is enjoined against such acts by this Court.

SECOND CAUSE OF ACTION

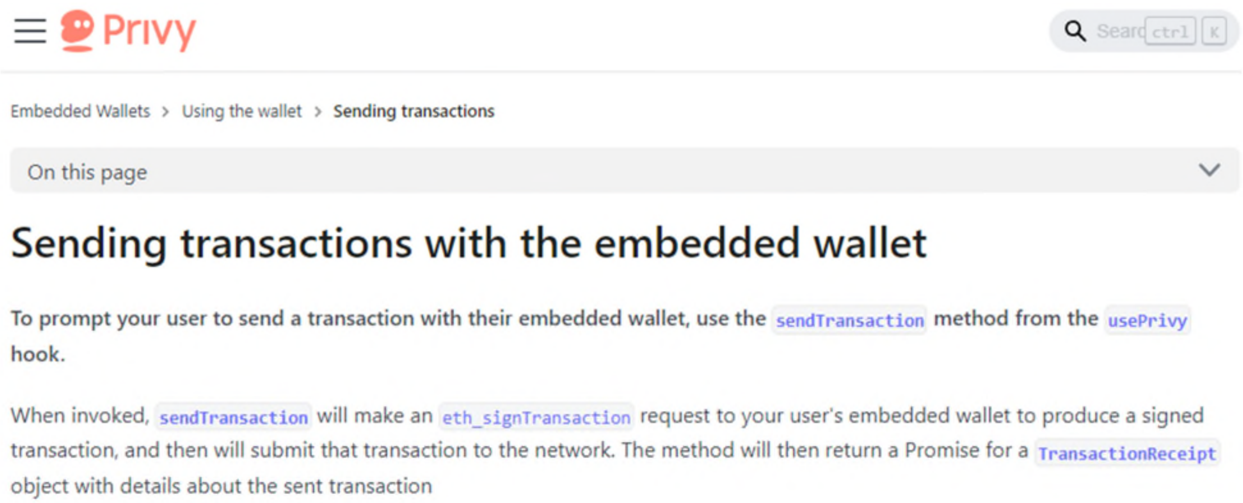
(INFRINGEMENT OF U.S. PATENT NO. 11,818,120)

65. Magic incorporates by reference and re-states paragraphs 1 through 64.

66. Magic is the owner of all right, title, and interest in and to the ’120 Patent.

67. The Accused System infringes the ’120 Patent. For example, the Accused System meets each limitation of at least claims 1, 6, and 9 of the ’120 Patent.

68. To the extent the preamble is limiting, the Accused System implements “[a] method for signing transaction data for a decentralized application transaction.” ’120 Patent, claim 1. The Accused System, for instance, “produce[s] a signed transaction” in the context of decentralized application transactions:



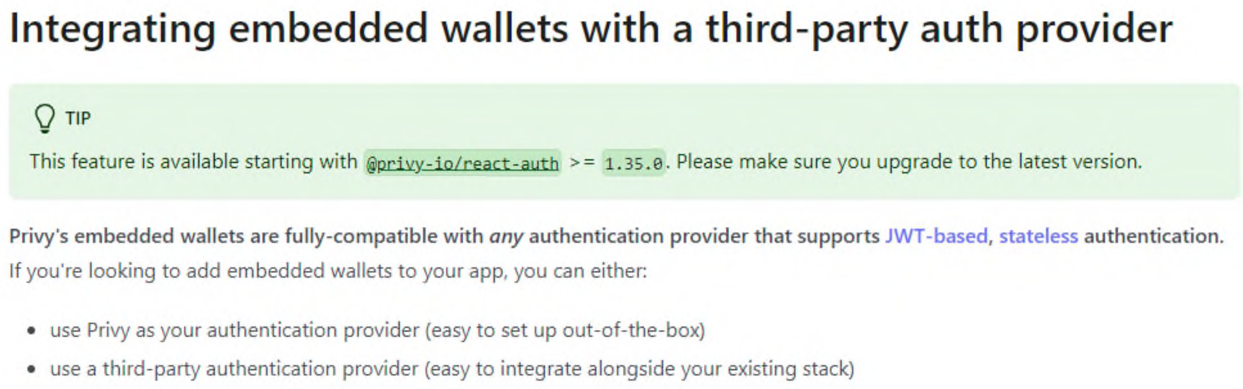
Sending transactions with the embedded wallet

To prompt your user to send a transaction with their embedded wallet, use the `sendTransaction` method from the `usePrivy` hook.

When invoked, `sendTransaction` will make an `eth_signTransaction` request to your user's embedded wallet to produce a signed transaction, and then will submit that transaction to the network. The method will then return a Promise for a `TransactionReceipt` object with details about the sent transaction

https://docs.privy.io/guide/frontend/embedded/usage/transacting.

69. The Accused System signs transaction data with a method comprising "receiving, by a first computing environment, an access token that corresponds to a user authentication by a second computing environment, to sign transaction data for a decentralized application transaction." '120 Patent, claim 1.



Integrating embedded wallets with a third-party auth provider

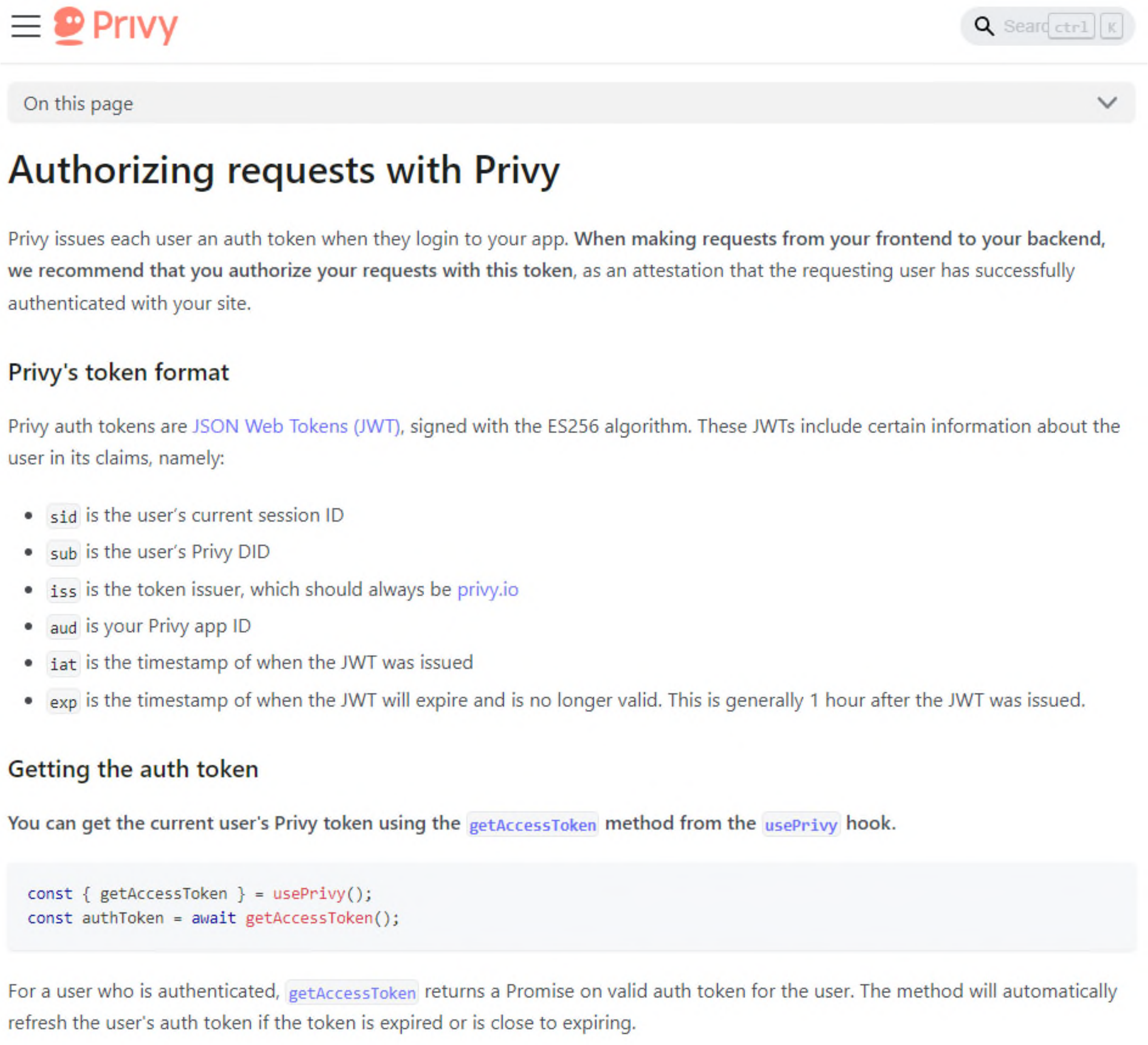
TIP

This feature is available starting with `@privy-io/react-auth` >= `1.35.0`. Please make sure you upgrade to the latest version.

Privy's embedded wallets are fully-compatible with *any* authentication provider that supports JWT-based, stateless authentication. If you're looking to add embedded wallets to your app, you can either:

- use Privy as your authentication provider (easy to set up out-of-the-box)
- use a third-party authentication provider (easy to integrate alongside your existing stack)

https://docs.privy.io/guide/guides/third-party-auth.



Authorizing requests with Privy

Privy issues each user an auth token when they login to your app. **When making requests from your frontend to your backend, we recommend that you authorize your requests with this token**, as an attestation that the requesting user has successfully authenticated with your site.

Privy's token format

Privy auth tokens are JSON Web Tokens (JWT), signed with the ES256 algorithm. These JWTs include certain information about the user in its claims, namely:

- `sid` is the user's current session ID
- `sub` is the user's Privy DID
- `iss` is the token issuer, which should always be privy.io
- `aud` is your Privy app ID
- `iat` is the timestamp of when the JWT was issued
- `exp` is the timestamp of when the JWT will expire and is no longer valid. This is generally 1 hour after the JWT was issued.

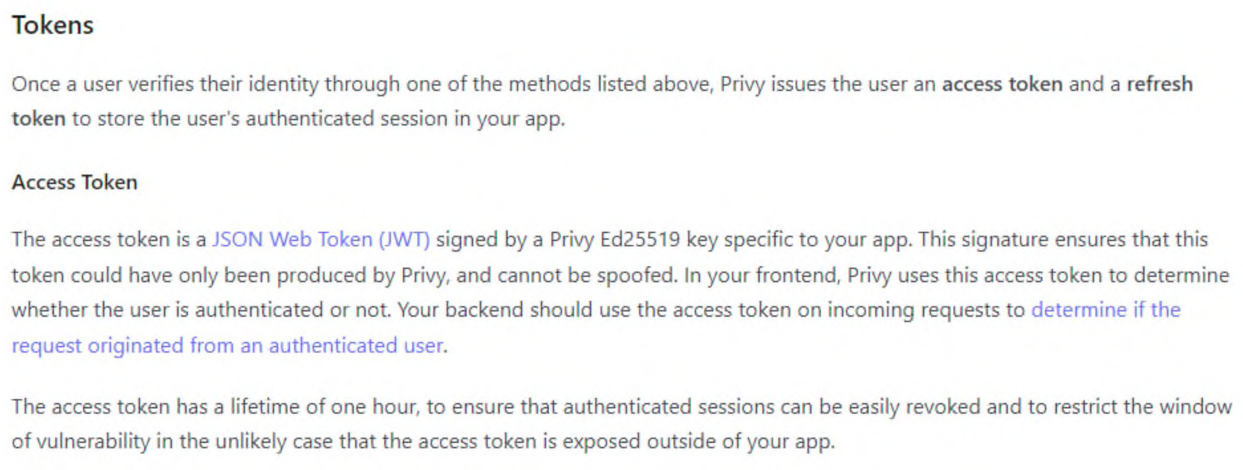
Getting the auth token

You can get the current user's Privy token using the `getAccessToken` method from the `usePrivy` hook.

```
const { getAccessToken } = usePrivy();
const authToken = await getAccessToken();
```

For a user who is authenticated, `getAccessToken` returns a Promise on valid auth token for the user. The method will automatically refresh the user's auth token if the token is expired or is close to expiring.

https://docs.privy.io/guide/frontend/authorization.



Tokens

Once a user verifies their identity through one of the methods listed above, Privy issues the user an **access token** and a **refresh token** to store the user's authenticated session in your app.

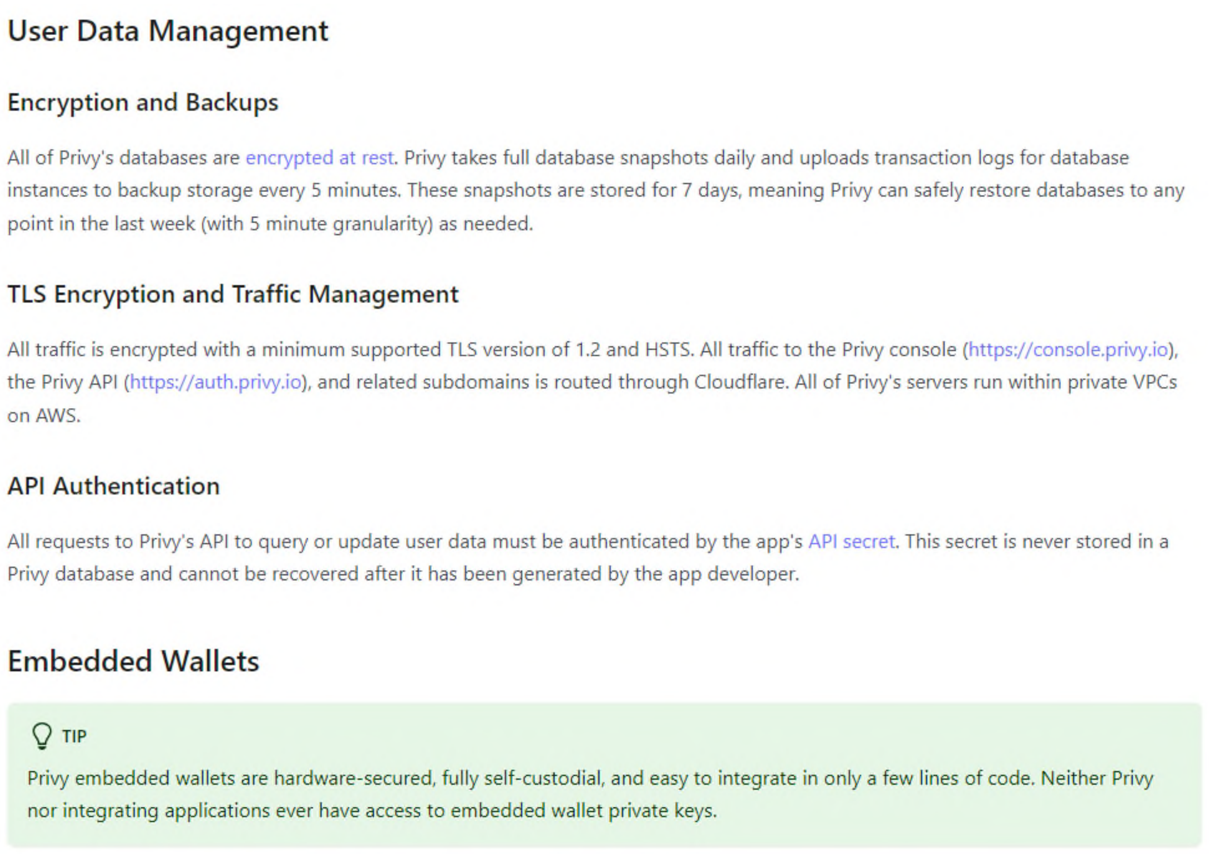
Access Token

The access token is a JSON Web Token (JWT) signed by a Privy Ed25519 key specific to your app. This signature ensures that this token could have only been produced by Privy, and cannot be spoofed. In your frontend, Privy uses this access token to determine whether the user is authenticated or not. Your backend should use the access token on incoming requests to determine if the request originated from an authenticated user.

The access token has a lifetime of one hour, to ensure that authenticated sessions can be easily revoked and to restrict the window of vulnerability in the unlikely case that the access token is exposed outside of your app.

https://docs.privy.io/guide/security.

70. The Accused System also operates by "sending a request bypassing the second computing environment, by the first computing environment to access a decrypted version of a private key information from a third party security system, wherein the private key information is encrypted using the third party security system" and "receiving, in response to the request, the decrypted version of the private key information at the first computing environment from the third party security system." '120 Patent, claim 1.



User Data Management

Encryption and Backups

All of Privy's databases are encrypted at rest. Privy takes full database snapshots daily and uploads transaction logs for database instances to backup storage every 5 minutes. These snapshots are stored for 7 days, meaning Privy can safely restore databases to any point in the last week (with 5 minute granularity) as needed.

TLS Encryption and Traffic Management

All traffic is encrypted with a minimum supported TLS version of 1.2 and HSTS. All traffic to the Privy console (https://console.privy.io), the Privy API (https://auth.privy.io), and related subdomains is routed through Cloudflare. All of Privy's servers run within private VPCs on AWS.

API Authentication

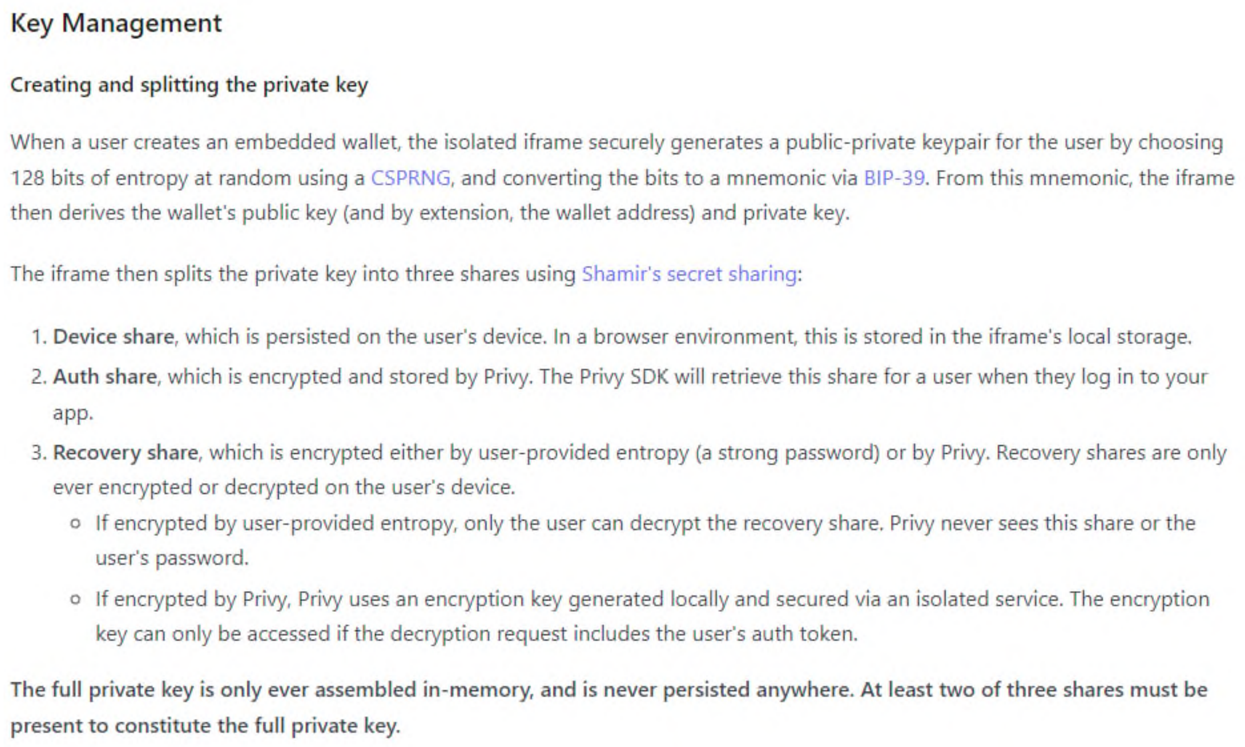
All requests to Privy's API to query or update user data must be authenticated by the app's API secret. This secret is never stored in a Privy database and cannot be recovered after it has been generated by the app developer.

Embedded Wallets

TIP

Privy embedded wallets are hardware-secured, fully self-custodial, and easy to integrate in only a few lines of code. Neither Privy nor integrating applications ever have access to embedded wallet private keys.

https://docs.privy.io/guide/security#user-data-management.



Key Management

Creating and splitting the private key

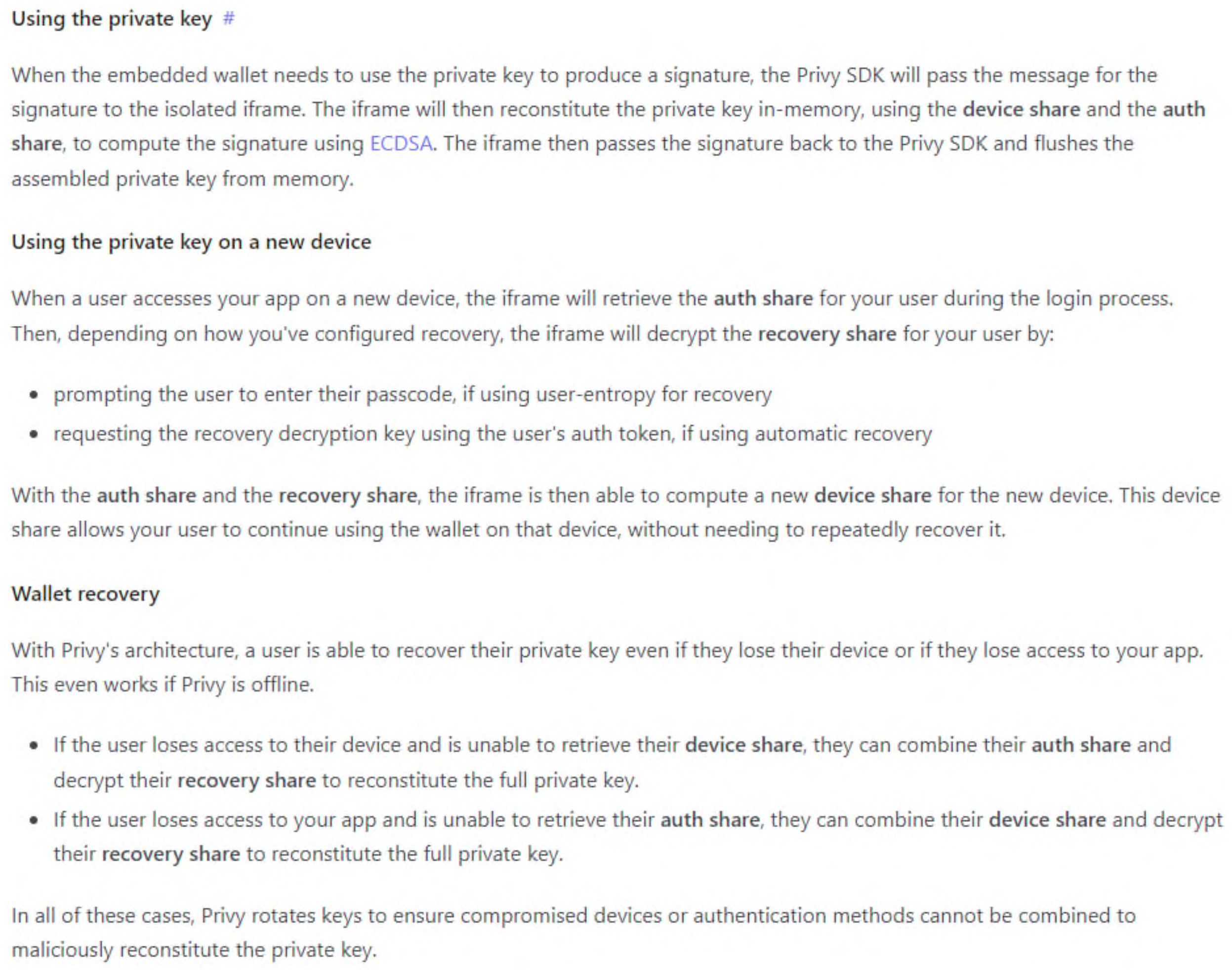
When a user creates an embedded wallet, the isolated iframe securely generates a public-private keypair for the user by choosing 128 bits of entropy at random using a CSPRNG, and converting the bits to a mnemonic via BIP-39. From this mnemonic, the iframe then derives the wallet's public key (and by extension, the wallet address) and private key.

The iframe then splits the private key into three shares using Shamir's secret sharing:

1. **Device share**, which is persisted on the user's device. In a browser environment, this is stored in the iframe's local storage.
2. **Auth share**, which is encrypted and stored by Privy. The Privy SDK will retrieve this share for a user when they log in to your app.
3. **Recovery share**, which is encrypted either by user-provided entropy (a strong password) or by Privy. Recovery shares are only ever encrypted or decrypted on the user's device.
    - If encrypted by user-provided entropy, only the user can decrypt the recovery share. Privy never sees this share or the user's password.
    - If encrypted by Privy, Privy uses an encryption key generated locally and secured via an isolated service. The encryption key can only be accessed if the decryption request includes the user's auth token.

The full private key is only ever assembled in-memory, and is never persisted anywhere. At least two of three shares must be present to constitute the full private key.

https://docs.privy.io/guide/security#user-data-management.



Using the private key #

When the embedded wallet needs to use the private key to produce a signature, the Privy SDK will pass the message for the signature to the isolated iframe. The iframe will then reconstitute the private key in-memory, using the **device share** and the **auth share**, to compute the signature using ECDSA. The iframe then passes the signature back to the Privy SDK and flushes the assembled private key from memory.

Using the private key on a new device

When a user accesses your app on a new device, the iframe will retrieve the **auth share** for your user during the login process. Then, depending on how you've configured recovery, the iframe will decrypt the **recovery share** for your user by:

- prompting the user to enter their passcode, if using user-entropy for recovery
- requesting the recovery decryption key using the user's auth token, if using automatic recovery

With the **auth share** and the **recovery share**, the iframe is then able to compute a new **device share** for the new device. This device share allows your user to continue using the wallet on that device, without needing to repeatedly recover it.

Wallet recovery

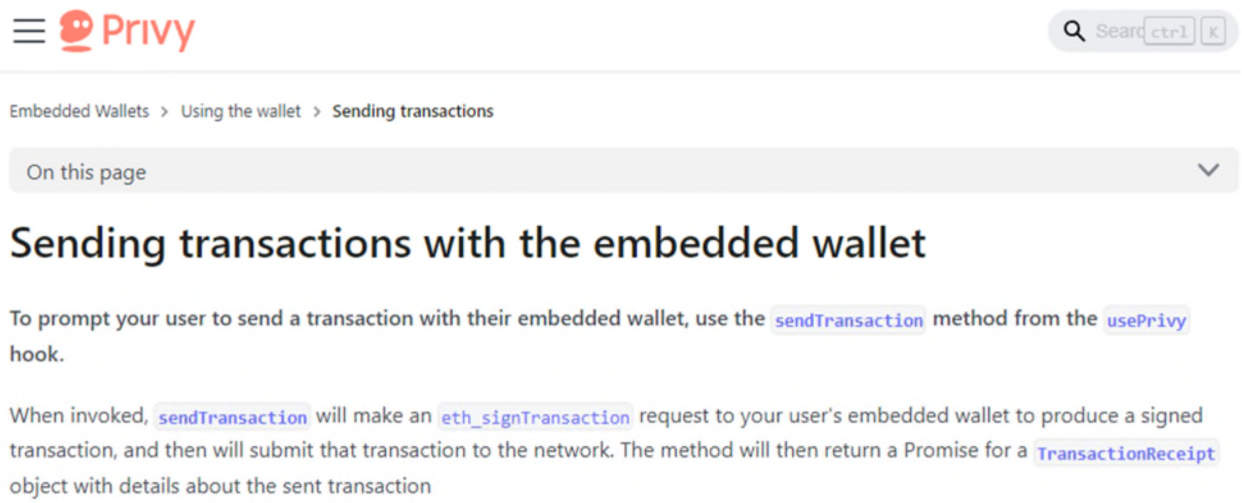
With Privy's architecture, a user is able to recover their private key even if they lose their device or if they lose access to your app. This even works if Privy is offline.

- If the user loses access to their device and is unable to retrieve their **device share**, they can combine their **auth share** and decrypt their **recovery share** to reconstitute the full private key.
- If the user loses access to your app and is unable to retrieve their **auth share**, they can combine their **device share** and decrypt their **recovery share** to reconstitute the full private key.

In all of these cases, Privy rotates keys to ensure compromised devices or authentication methods cannot be combined to maliciously reconstitute the private key.

Id.

71. The Accused System further operates by “signing, by the first computing environment, the transaction data with the decrypted version of the private key information.” ’120 Patent, claim 1. This is illustrated by the images in Paragraph 70 above, which show that the Accused System signs transaction data with the decrypted versions of the key shares that it uses. Other portions of Privy’s website make this clear as well:



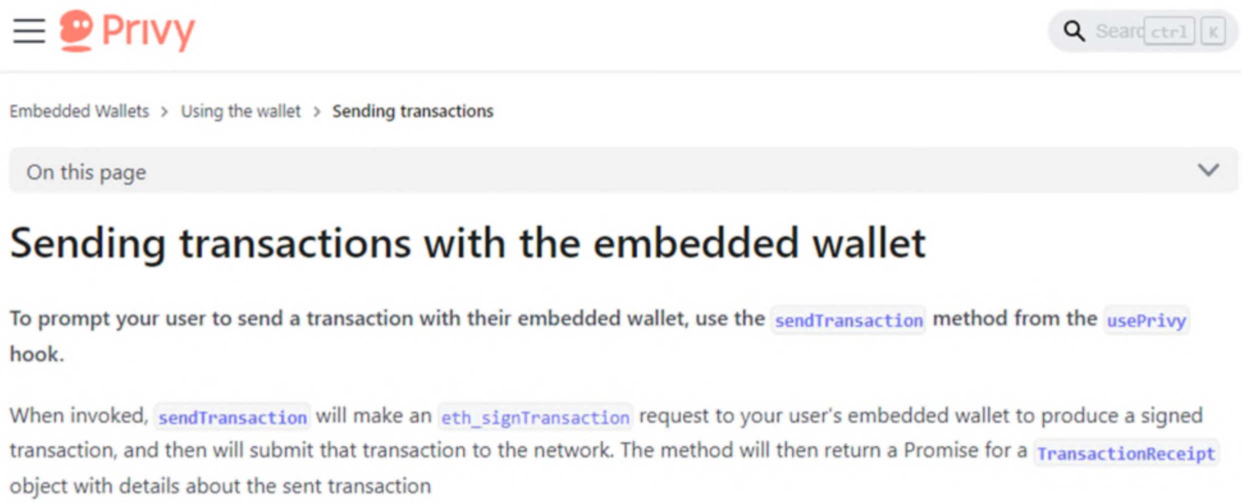
Sending transactions with the embedded wallet

To prompt your user to send a transaction with their embedded wallet, use the `sendTransaction` method from the `usePrivy` hook.

When invoked, `sendTransaction` will make an `eth_signTransaction` request to your user's embedded wallet to produce a signed transaction, and then will submit that transaction to the network. The method will then return a Promise for a `TransactionReceipt` object with details about the sent transaction

https://docs.privy.io/guide/frontend/embedded/usage/transacting.

72. The Accused System also operates by "sending, by the first computing environment, the signed transaction data for submission to a decentralized network; wherein the third party security system includes an access token service; wherein sending the request, bypassing the second computing environment, by the first computing environment to access a decrypted version of the private key information includes sending a message that includes the access token by the first computing environment to the access token service." '120 Patent, claim 1. For example, Privy's Accused System sends signed transaction data to a decentralized blockchain network and works with a third-party security system that includes an access token service.



Embedded Wallets > Using the wallet > **Sending transactions**

On this page

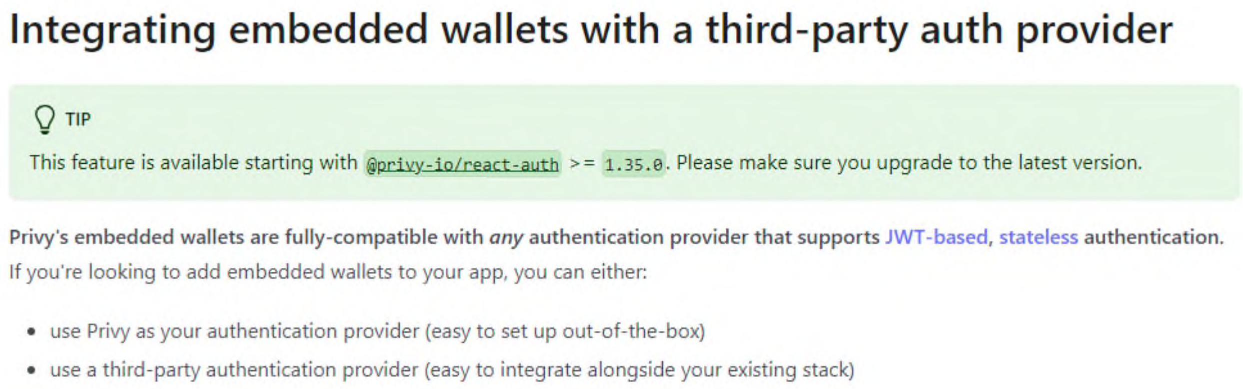
Sending transactions with the embedded wallet

To prompt your user to send a transaction with their embedded wallet, use the `sendTransaction` method from the `usePrivy` hook.

When invoked, `sendTransaction` will make an `eth_signTransaction` request to your user's embedded wallet to produce a signed transaction, and then will submit that transaction to the network. The method will then return a Promise for a `TransactionReceipt` object with details about the sent transaction

https://docs.privy.io/guide/frontend/embedded/usage/transacting.

73. When Privy sends requests to access a decrypted version of the private keys, the request includes a message that contains the access token.



Integrating embedded wallets with a third-party auth provider

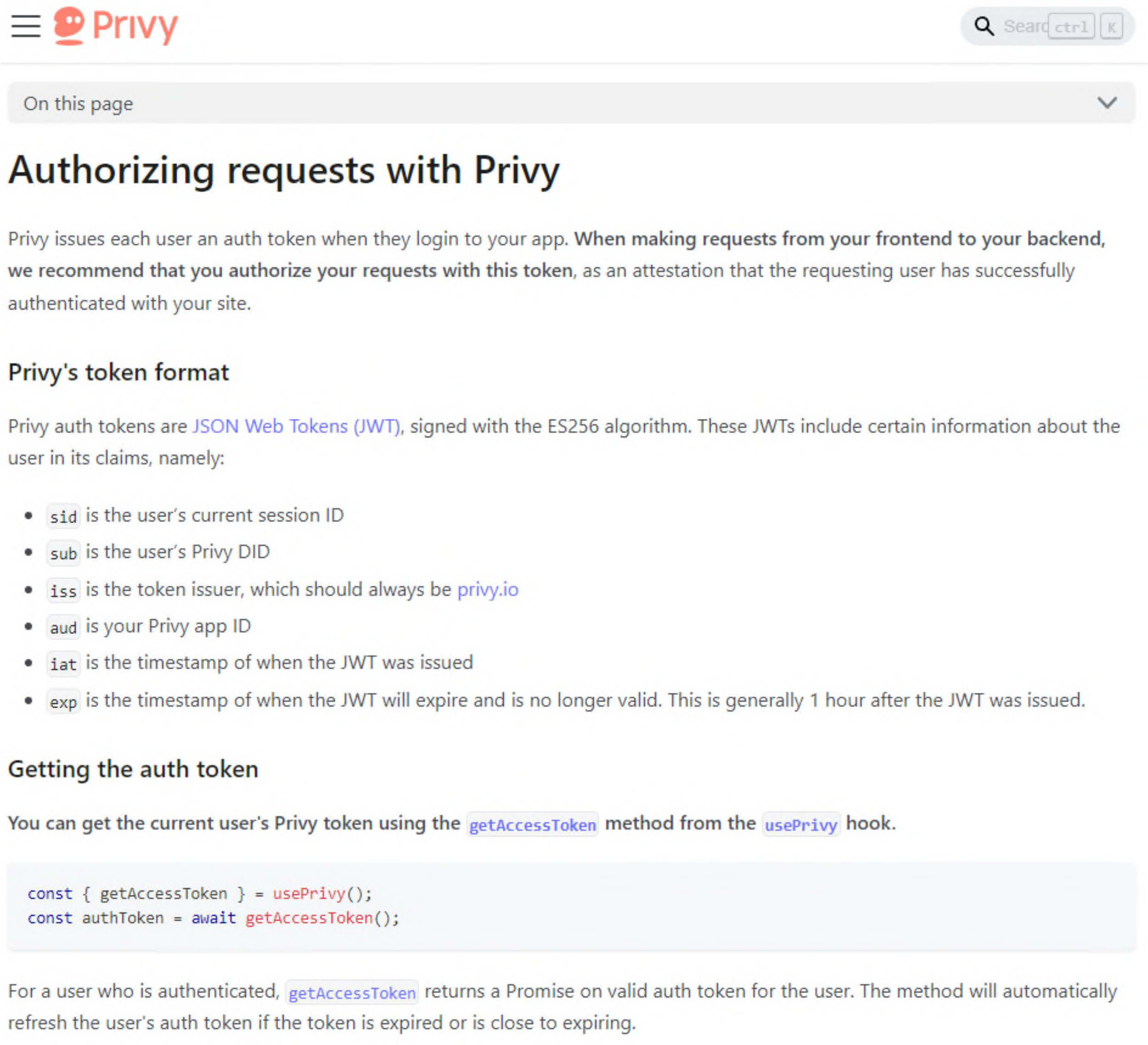
TIP

This feature is available starting with `@privy-io/react-auth` >= `1.35.0`. Please make sure you upgrade to the latest version.

Privy's embedded wallets are fully-compatible with *any* authentication provider that supports JWT-based, stateless authentication. If you're looking to add embedded wallets to your app, you can either:

- use Privy as your authentication provider (easy to set up out-of-the-box)
- use a third-party authentication provider (easy to integrate alongside your existing stack)

https://docs.privy.io/guide/guides/third-party-auth.



Authorizing requests with Privy

Privy issues each user an auth token when they login to your app. **When making requests from your frontend to your backend, we recommend that you authorize your requests with this token**, as an attestation that the requesting user has successfully authenticated with your site.

Privy's token format

Privy auth tokens are JSON Web Tokens (JWT), signed with the ES256 algorithm. These JWTs include certain information about the user in its claims, namely:

- `sid` is the user's current session ID
- `sub` is the user's Privy DID
- `iss` is the token issuer, which should always be privy.io
- `aud` is your Privy app ID
- `iat` is the timestamp of when the JWT was issued
- `exp` is the timestamp of when the JWT will expire and is no longer valid. This is generally 1 hour after the JWT was issued.

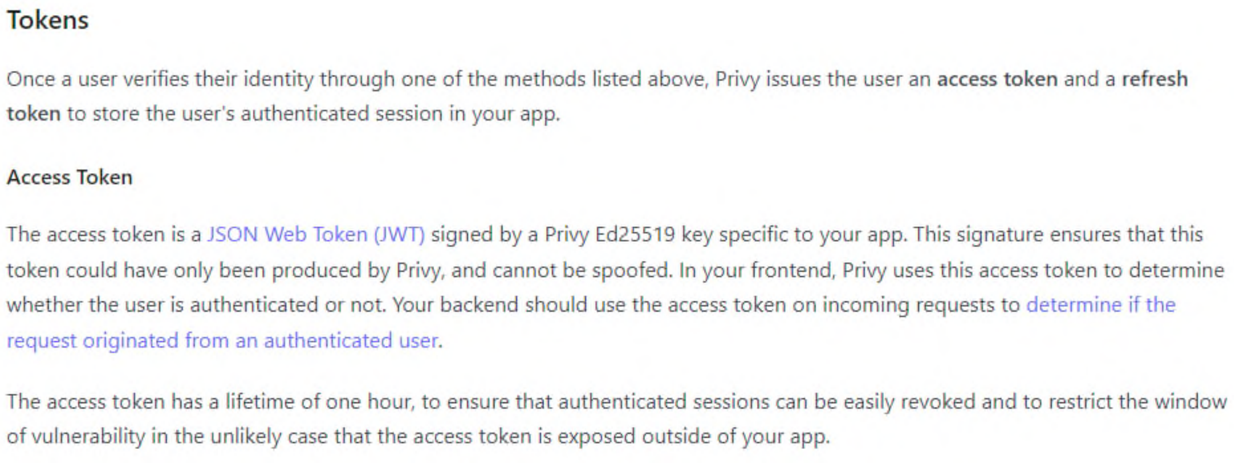
Getting the auth token

You can get the current user's Privy token using the `getAccessToken` method from the `usePrivy` hook.

```
const { getAccessToken } = usePrivy();
const authToken = await getAccessToken();
```

For a user who is authenticated, `getAccessToken` returns a Promise on valid auth token for the user. The method will automatically refresh the user's auth token if the token is expired or is close to expiring.

https://docs.privy.io/guide/frontend/authorization.



Tokens

Once a user verifies their identity through one of the methods listed above, Privy issues the user an **access token** and a **refresh token** to store the user's authenticated session in your app.

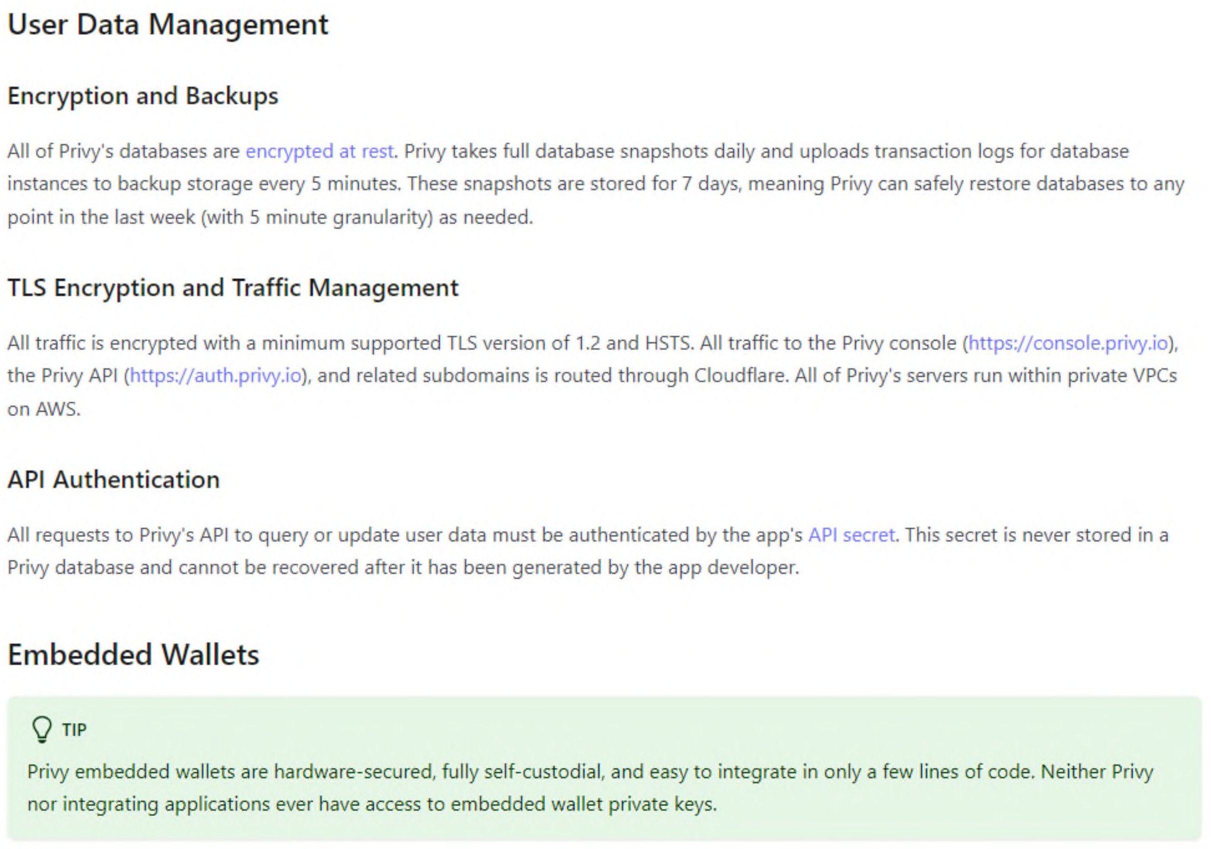
Access Token

The access token is a JSON Web Token (JWT) signed by a Privy Ed25519 key specific to your app. This signature ensures that this token could have only been produced by Privy, and cannot be spoofed. In your frontend, Privy uses this access token to determine whether the user is authenticated or not. Your backend should use the access token on incoming requests to determine if the request originated from an authenticated user.

The access token has a lifetime of one hour, to ensure that authenticated sessions can be easily revoked and to restrict the window of vulnerability in the unlikely case that the access token is exposed outside of your app.

https://docs.privy.io/guide/security.

74. In addition, the Accused System operates by using a "third party security system [that] includes a hardware security module." '120 Patent, claim 6.



User Data Management

Encryption and Backups

All of Privy's databases are encrypted at rest. Privy takes full database snapshots daily and uploads transaction logs for database instances to backup storage every 5 minutes. These snapshots are stored for 7 days, meaning Privy can safely restore databases to any point in the last week (with 5 minute granularity) as needed.

TLS Encryption and Traffic Management

All traffic is encrypted with a minimum supported TLS version of 1.2 and HSTS. All traffic to the Privy console (https://console.privy.io), the Privy API (https://auth.privy.io), and related subdomains is routed through Cloudflare. All of Privy's servers run within private VPCs on AWS.

API Authentication

All requests to Privy's API to query or update user data must be authenticated by the app's API secret. This secret is never stored in a Privy database and cannot be recovered after it has been generated by the app developer.

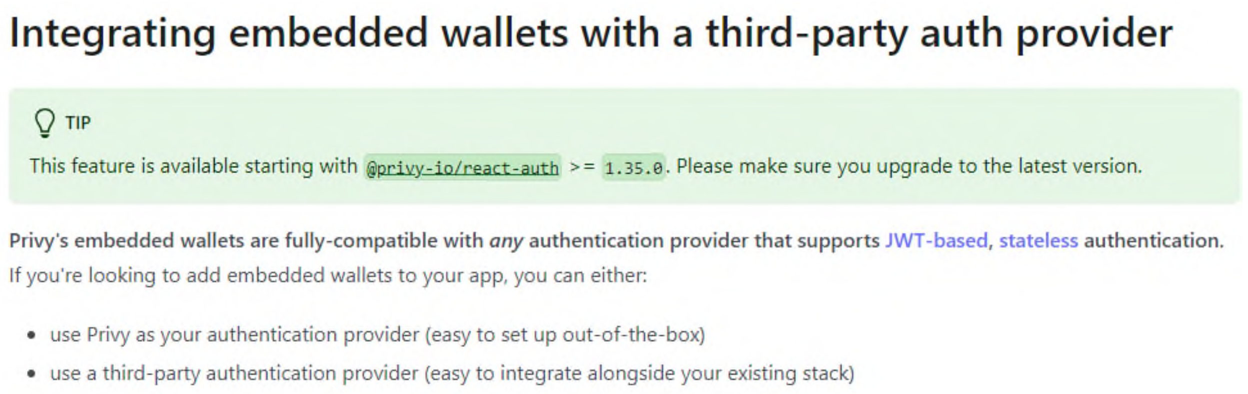
Embedded Wallets

TIP

Privy embedded wallets are hardware-secured, fully self-custodial, and easy to integrate in only a few lines of code. Neither Privy nor integrating applications ever have access to embedded wallet private keys.

https://docs.privy.io/guide/security#user-data-management.

75. The Accused System further operates by "receiving the access token at the first computing environment," wherein doing so "includes receiving the access token from the access token service." '120 Patent, claim 6.



Integrating embedded wallets with a third-party auth provider

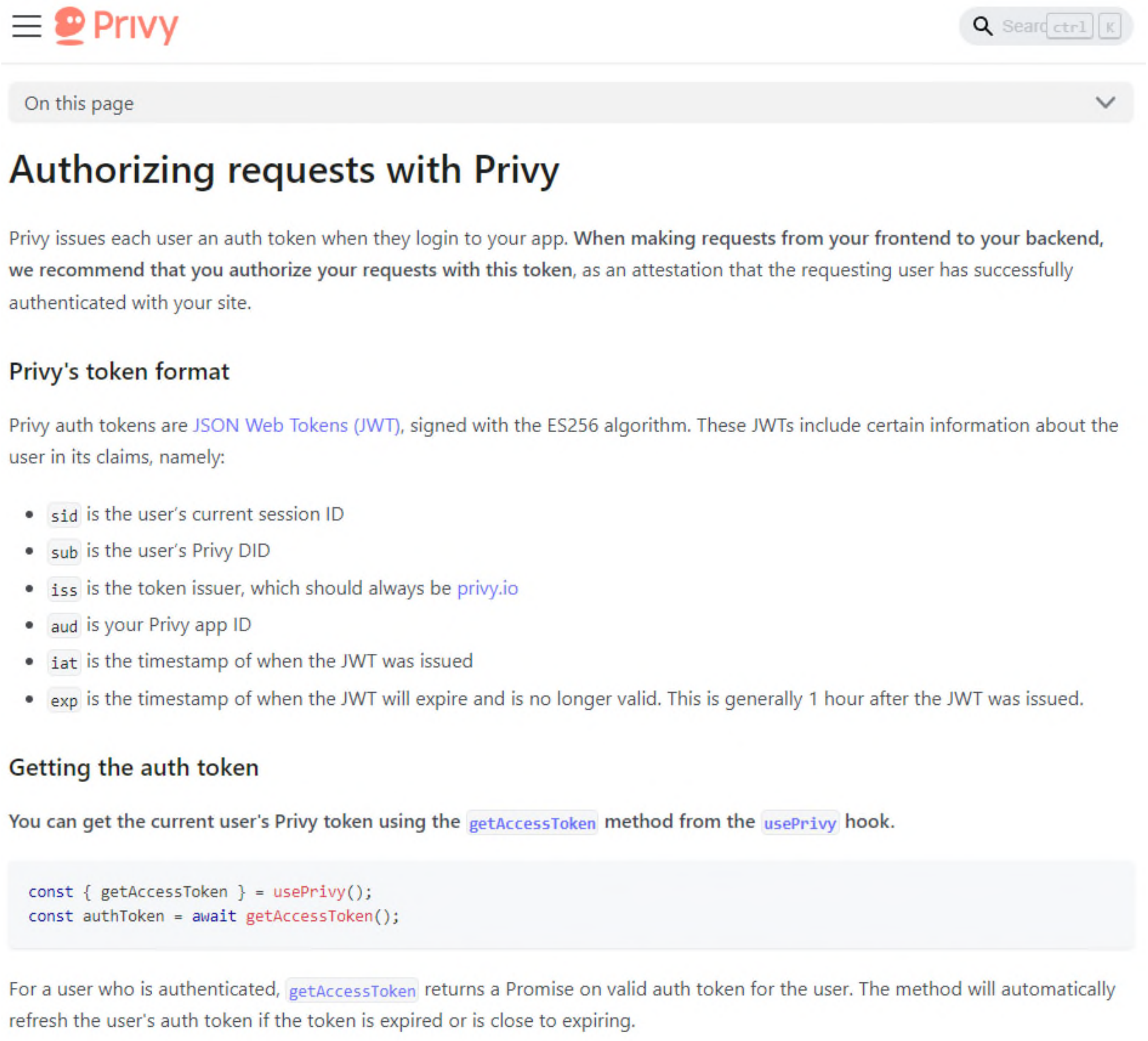
TIP

This feature is available starting with `@privy-io/react-auth` >= `1.35.0`. Please make sure you upgrade to the latest version.

Privy's embedded wallets are fully-compatible with *any* authentication provider that supports JWT-based, stateless authentication. If you're looking to add embedded wallets to your app, you can either:

- use Privy as your authentication provider (easy to set up out-of-the-box)
- use a third-party authentication provider (easy to integrate alongside your existing stack)

https://docs.privy.io/guide/guides/third-party-auth.



Authorizing requests with Privy

Privy issues each user an auth token when they login to your app. **When making requests from your frontend to your backend, we recommend that you authorize your requests with this token**, as an attestation that the requesting user has successfully authenticated with your site.

Privy's token format

Privy auth tokens are JSON Web Tokens (JWT), signed with the ES256 algorithm. These JWTs include certain information about the user in its claims, namely:

- `sid` is the user's current session ID
- `sub` is the user's Privy DID
- `iss` is the token issuer, which should always be privy.io
- `aud` is your Privy app ID
- `iat` is the timestamp of when the JWT was issued
- `exp` is the timestamp of when the JWT will expire and is no longer valid. This is generally 1 hour after the JWT was issued.

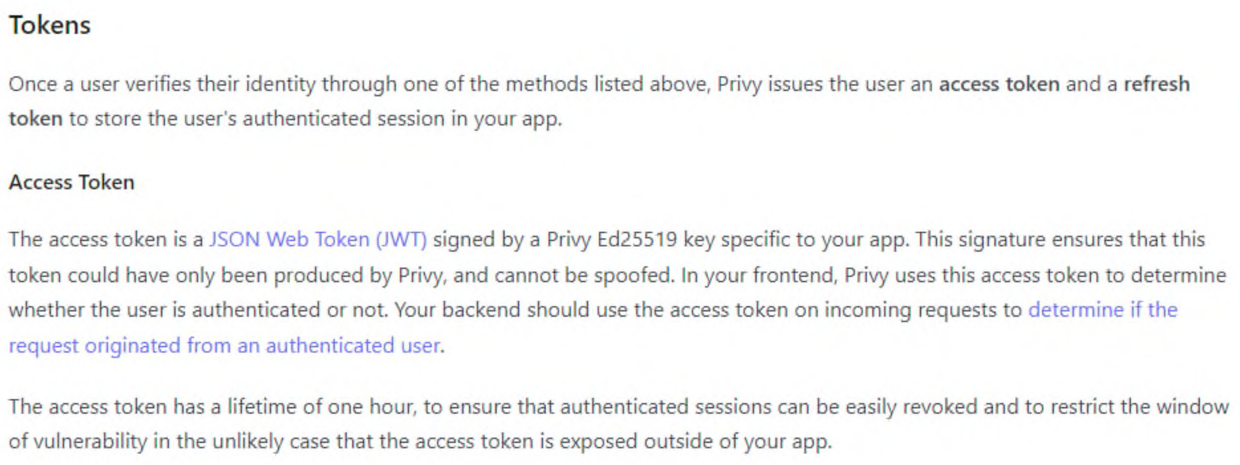
Getting the auth token

You can get the current user's Privy token using the `getAccessToken` method from the `usePrivy` hook.

```
const { getAccessToken } = usePrivy();
const authToken = await getAccessToken();
```

For a user who is authenticated, `getAccessToken` returns a Promise on valid auth token for the user. The method will automatically refresh the user's auth token if the token is expired or is close to expiring.

https://docs.privy.io/guide/frontend/authorization.



Tokens

Once a user verifies their identity through one of the methods listed above, Privy issues the user an **access token** and a **refresh token** to store the user's authenticated session in your app.

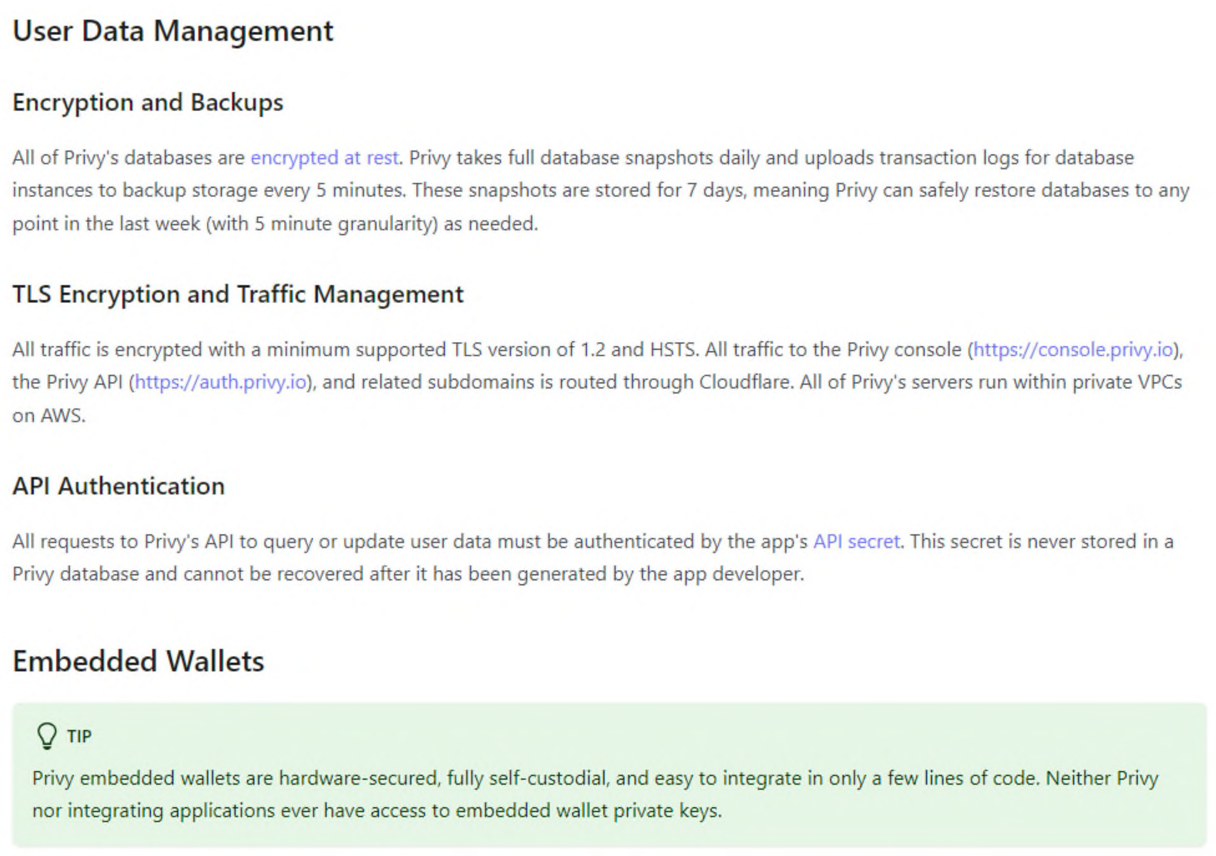
Access Token

The access token is a JSON Web Token (JWT) signed by a Privy Ed25519 key specific to your app. This signature ensures that this token could have only been produced by Privy, and cannot be spoofed. In your frontend, Privy uses this access token to determine whether the user is authenticated or not. Your backend should use the access token on incoming requests to determine if the request originated from an authenticated user.

The access token has a lifetime of one hour, to ensure that authenticated sessions can be easily revoked and to restrict the window of vulnerability in the unlikely case that the access token is exposed outside of your app.

https://docs.privy.io/guide/security.

76. The Accused System also operates by "sending the request, bypassing the second computing environment, by the first by the first computing environment to access a decrypted version of the private key information," wherein doing so "includes sending a message that includes a scoped credential, by the first computing environment to the hardware security module [HSM]." '120 Patent, claim 6. As illustrated below, the Accused System allows for the client network to send a request to encrypt the key, which is called the "Recovery Share," to a third-party HSM. The Accused System also provides the client with the ability to retrieve the key and decrypt it from the HSM. It does so through a request that "bypass[es] the second computing environment" (Privy's infrastructure) "by the first computing environment" (the user's client-facing browser) "to access a decrypted version of the private key information" following "sending a message" "by the first computing environment to the [HSM]." This request "includes a scoped credential."



User Data Management

Encryption and Backups

All of Privy's databases are encrypted at rest. Privy takes full database snapshots daily and uploads transaction logs for database instances to backup storage every 5 minutes. These snapshots are stored for 7 days, meaning Privy can safely restore databases to any point in the last week (with 5 minute granularity) as needed.

TLS Encryption and Traffic Management

All traffic is encrypted with a minimum supported TLS version of 1.2 and HSTS. All traffic to the Privy console (https://console.privy.io), the Privy API (https://auth.privy.io), and related subdomains is routed through Cloudflare. All of Privy's servers run within private VPCs on AWS.

API Authentication

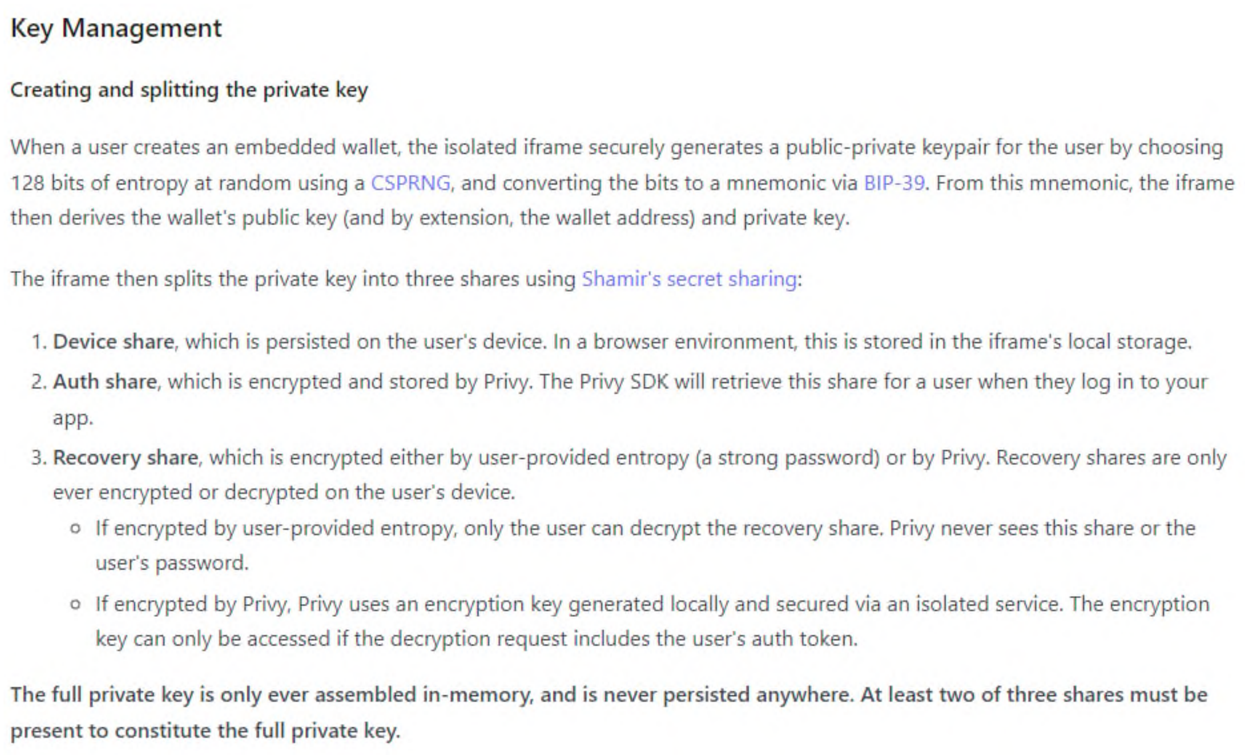
All requests to Privy's API to query or update user data must be authenticated by the app's API secret. This secret is never stored in a Privy database and cannot be recovered after it has been generated by the app developer.

Embedded Wallets

TIP

Privy embedded wallets are hardware-secured, fully self-custodial, and easy to integrate in only a few lines of code. Neither Privy nor integrating applications ever have access to embedded wallet private keys.

https://docs.privy.io/guide/security#user-data-management.



Key Management

Creating and splitting the private key

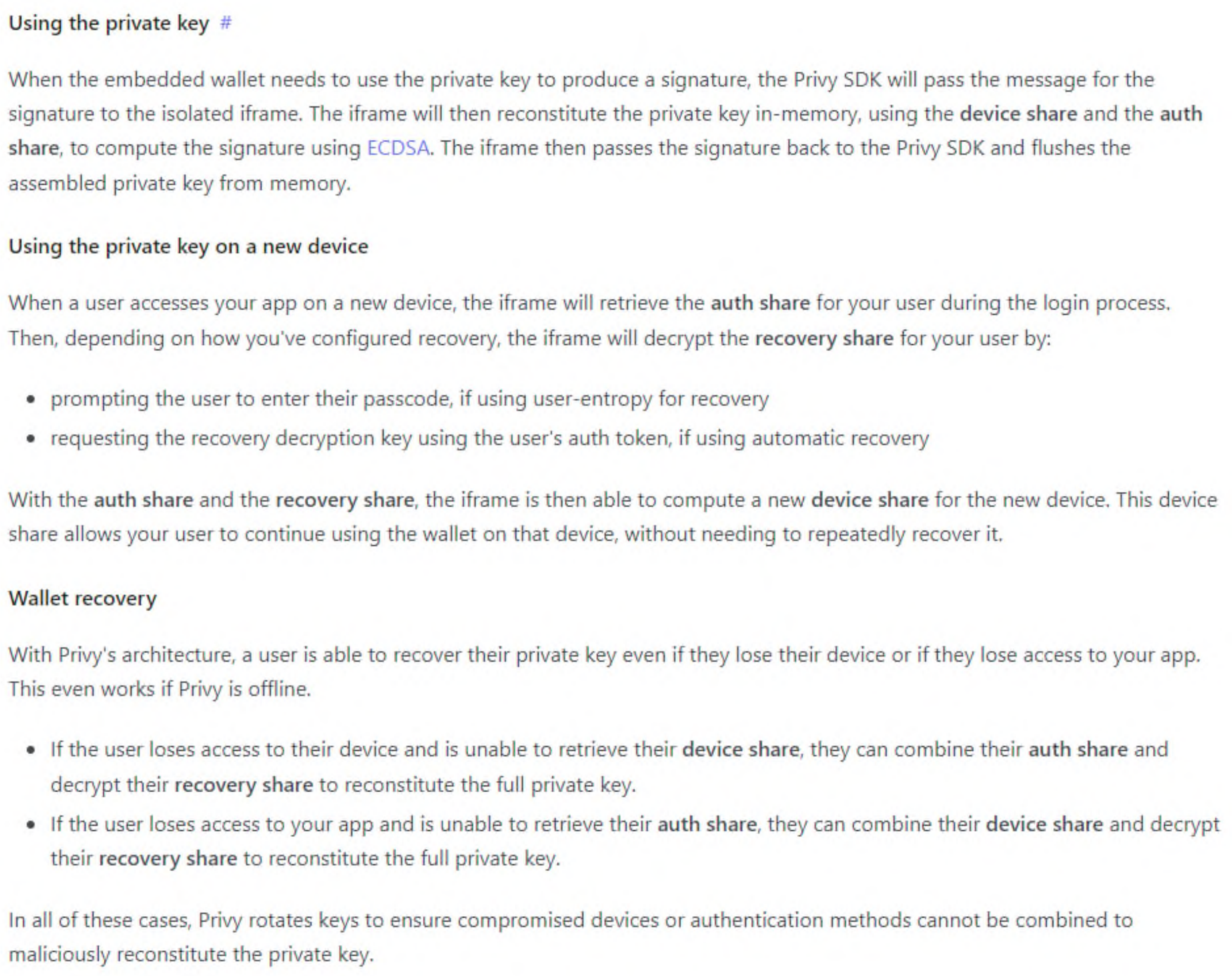
When a user creates an embedded wallet, the isolated iframe securely generates a public-private keypair for the user by choosing 128 bits of entropy at random using a CSPRNG, and converting the bits to a mnemonic via BIP-39. From this mnemonic, the iframe then derives the wallet's public key (and by extension, the wallet address) and private key.

The iframe then splits the private key into three shares using Shamir's secret sharing:

1. **Device share**, which is persisted on the user's device. In a browser environment, this is stored in the iframe's local storage.
2. **Auth share**, which is encrypted and stored by Privy. The Privy SDK will retrieve this share for a user when they log in to your app.
3. **Recovery share**, which is encrypted either by user-provided entropy (a strong password) or by Privy. Recovery shares are only ever encrypted or decrypted on the user's device.
   - If encrypted by user-provided entropy, only the user can decrypt the recovery share. Privy never sees this share or the user's password.
   - If encrypted by Privy, Privy uses an encryption key generated locally and secured via an isolated service. The encryption key can only be accessed if the decryption request includes the user's auth token.

The full private key is only ever assembled in-memory, and is never persisted anywhere. At least two of three shares must be present to constitute the full private key.

https://docs.privy.io/guide/security#user-data-management.



Using the private key

When the embedded wallet needs to use the private key to produce a signature, the Privy SDK will pass the message for the signature to the isolated iframe. The iframe will then reconstitute the private key in-memory, using the **device share** and the **auth share**, to compute the signature using ECDSA. The iframe then passes the signature back to the Privy SDK and flushes the assembled private key from memory.

Using the private key on a new device

When a user accesses your app on a new device, the iframe will retrieve the **auth share** for your user during the login process. Then, depending on how you've configured recovery, the iframe will decrypt the **recovery share** for your user by:

- prompting the user to enter their passcode, if using user-entropy for recovery
- requesting the recovery decryption key using the user's auth token, if using automatic recovery

With the **auth share** and the **recovery share**, the iframe is then able to compute a new **device share** for the new device. This device share allows your user to continue using the wallet on that device, without needing to repeatedly recover it.

Wallet recovery

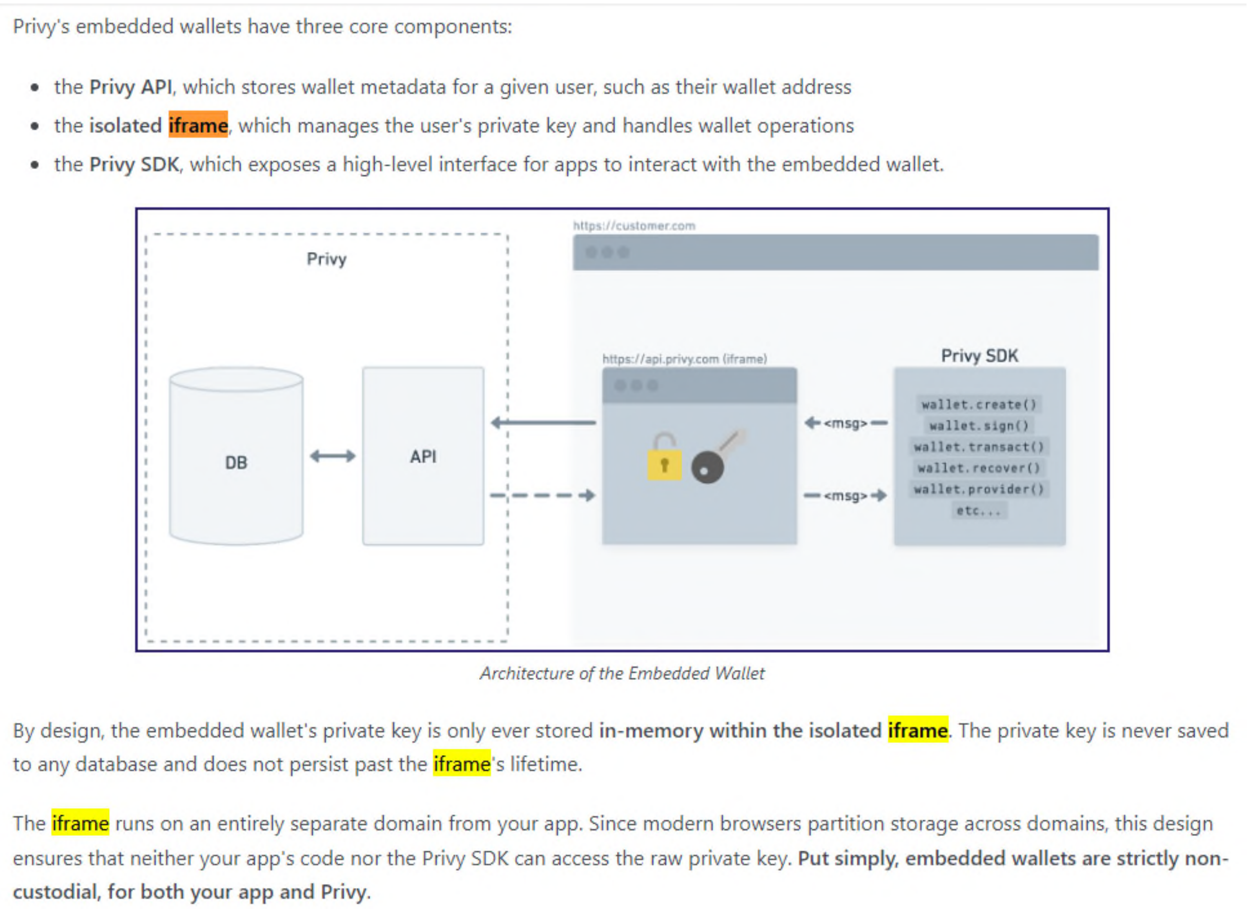
With Privy's architecture, a user is able to recover their private key even if they lose their device or if they lose access to your app. This even works if Privy is offline.

- If the user loses access to their device and is unable to retrieve their **device share**, they can combine their **auth share** and decrypt their **recovery share** to reconstitute the full private key.
- If the user loses access to your app and is unable to retrieve their **auth share**, they can combine their **device share** and decrypt their **recovery share** to reconstitute the full private key.

In all of these cases, Privy rotates keys to ensure compromised devices or authentication methods cannot be combined to maliciously reconstitute the private key.

Id.

77. Privy's Accused System further operates by "signing, by the by the first computing environment, the transaction data with the decrypted key information," wherein doing so "includes signing using the decrypted key information within an iframe at the first computing environment. '120 Patent, Claim 9.



Privy's embedded wallets have three core components:

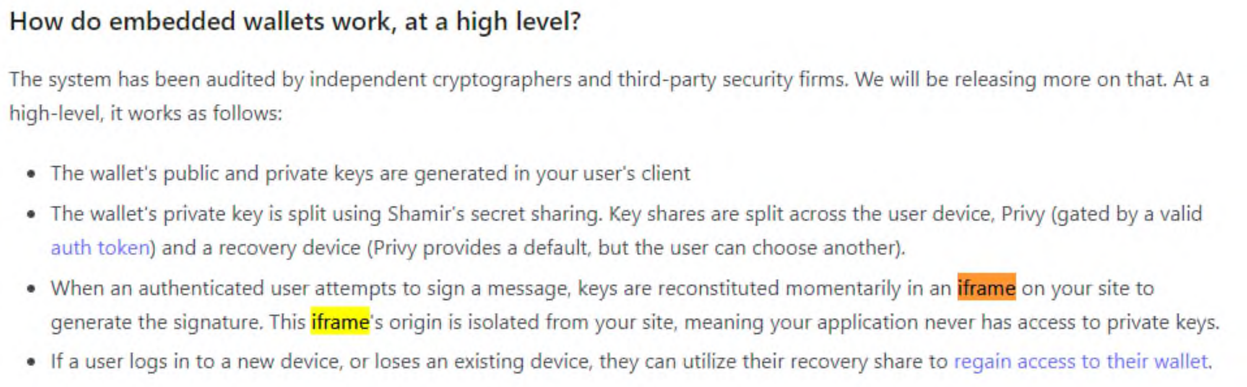
- the **Privy API**, which stores wallet metadata for a given user, such as their wallet address
- the **isolated iframe**, which manages the user's private key and handles wallet operations
- the **Privy SDK**, which exposes a high-level interface for apps to interact with the embedded wallet.

Architecture of the Embedded Wallet

By design, the embedded wallet's private key is only ever stored **in-memory within the isolated iframe**. The private key is never saved to any database and does not persist past the iframe's lifetime.

The iframe runs on an entirely separate domain from your app. Since modern browsers partition storage across domains, this design ensures that neither your app's code nor the Privy SDK can access the raw private key. **Put simply, embedded wallets are strictly non-custodial, for both your app and Privy.**

https://docs.privy.io/guide/security.



How do embedded wallets work, at a high level?

The system has been audited by independent cryptographers and third-party security firms. We will be releasing more on that. At a high-level, it works as follows:

- The wallet's public and private keys are generated in your user's client
- The wallet's private key is split using Shamir's secret sharing. Key shares are split across the user device, Privy (gated by a valid auth token) and a recovery device (Privy provides a default, but the user can choose another).
- When an authenticated user attempts to sign a message, keys are reconstituted momentarily in an iframe on your site to generate the signature. This iframe's origin is isolated from your site, meaning your application never has access to private keys.
- If a user logs in to a new device, or loses an existing device, they can utilize their recovery share to regain access to their wallet.

https://docs.privy.io/guide/frontend/embedded/faq.

78. Privy also has and continues to induce and/or contribute to the infringement of the ’120 Patent by inducing its customers to directly infringe the ’120 Patent (for example, when customers put into use the Accused System), and by contributing to such direct infringement.

79. Privy has known of the ’120 Patent and its infringement since it issued on November 14, 2023: on September 1, 2023, Magic sent Privy a letter notifying it of the application that issued as the ’120 Patent and Privy’s anticipated infringement of the patent once it issued.

80. As illustrated in paragraphs 68 to 77 above, Privy actively encourages its users to infringe claims of the ’120 Patent by providing the Accused System, advertising how the Accused System can be used in an infringing manner on its website and reference documents, and advertising and advising its users how to incorporate the Accused System into their applications in a manner that infringes the ’120 Patent.

81. On information and belief, the Accused System has no substantial non-infringing uses because the Accused System is specifically designed to infringe. Privy advertises on the home page of its website that one of the core features of the Accused System is that it allows for “beautiful authentication flows” and the ability to “bring[] web2-caliber UX—like sign in with email and social—to web3 products.” In other words, the Accused System allows users to more easily and securely sign up for a decentralized wallet. As explained in paragraphs 68 to 77 above, this core functionality of the Accused System is enabled by architecture that meets each element of at least claims 1, 6, and 9 of the ’120 Patent.

82. On information and belief, Privy will continue to infringe, induce infringement of, and/or contribute to infringement of the ’120 Patent unless enjoined by this Court.

83. As a result of Privy’s infringement of the ’120 Patent, Magic has suffered and will continue to suffer damages in an amount to be proven at trial.

84. As a result of Privy’s infringement of the ’120 Patent, Magic has suffered and will continue to suffer irreparable harm unless Privy is enjoined against such acts by this Court.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff Magic Labs, Inc. seeks relief against Defendant Horkos Inc. d/b/a Privy as follows:

a. for a judgment that Privy has directly infringed, either literally and/or under the doctrine of equivalents, induced infringement of, and/or contributed to infringement of one or more claims of the Asserted Patents in connection with the Accused Systems;

b. for a judgment and award of all damages sustained by Magic as a result of Privy's infringement, including supplemental damages for any continuing post-verdict infringement up until entry of the final judgment with an accounting as needed;

c. for a preliminary injunction enjoining Privy and anyone in concert with Privy from infringing, inducing infringement of, or contributing to the infringement of the Asserted Patents;

d. for a permanent injunction enjoining Privy and anyone in concert with Privy from infringing, inducing infringement of, or contributing to the infringement of the Asserted Patents;

e. for a judgment and an award of all interest and costs incurred; and

f. for a judgment and an award of such other and further relief as the Court may deem just and proper.

JURY DEMAND

Plaintiff Magic Labs, Inc. demands a trial by jury on all issues presented in the Complaint that are so triable.

DATED: December 22, 2023

OF COUNSEL:

Daralyn J. Durie
Ragesh K. Tangri
Timothy C. Saulsbury
Michael Burshteyn
Joyce C. Li
MORRISON & FOERSTER LLP
425 Market Street
San Francisco, CA 94105
(415) 268-7000

Sara Doudar
MORRISON & FOERSTER LLP
707 Wilshire Blvd., Suite 6000
Los Angeles, CA 90017
(213) 892-5200

MCCARTER & ENGLISH, LLP

/s/ Daniel M. Silver
Daniel M. Silver (#4758)
Alexandra M. Joyce (#6423)
Renaissance Centre
405 N. King Street, 8th Floor
Wilmington, DE 19801
T: (302) 984-6300
dsilver@mccarter.com
ajoyce@mccarter.com

Attorneys for Plaintiff
Magic Labs, Inc.