

CONF42

GOLANG

Securing Go APIs with Decentralized Identity Tokens



Mohammad Shahbaz Alam

Developer Advocate @ Magic

magic.link



@mdsbzalam

Agenda

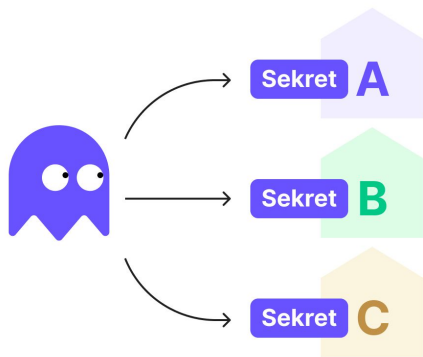
- User trust model of the Internet is broken.
- What is **Decentralized Identity Token**?
- **Auth** at Magic
- Build Go API
- Secure **Go API** with **Magic**



User trust model of the Internet is broken.

1: Users think of secrets / passwords specific to them to verify identity, and hand them off to apps owned by various companies.

59% of all users reuse their passwords across apps.



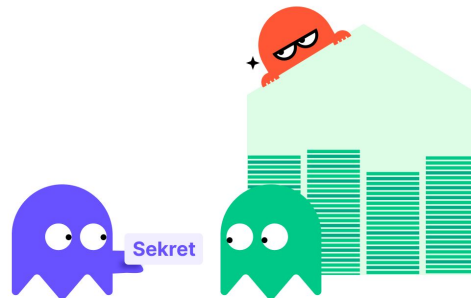
2: Users entrust companies to store their secrets securely and responsibly.

Many companies roll their own authentication with no prior knowledge on security at all.



3: Users access the companies' services by showing them the secret again which then gets matched with the stored secret.

Everytime the secret is shown, there's a risk of exposing it to hackers.

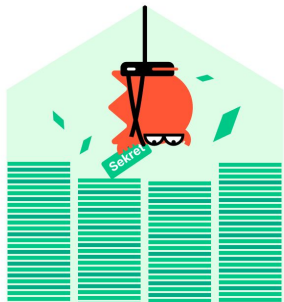




User trust model of the Internet is broken.

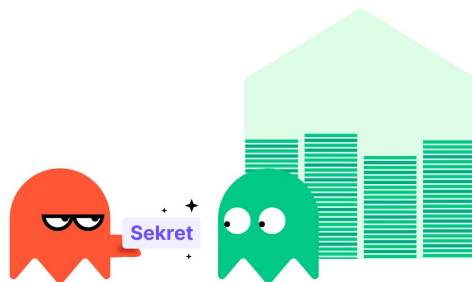
4: Companies might get hacked, and lose user secrets, along with users' trust.

48% of customers never come back after breach. The Equifax breach has cost them at least \$1.4B.



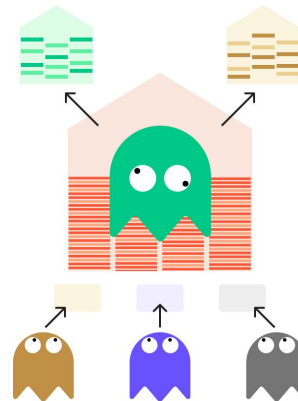
5: Hacker uses stolen secrets to impersonate users to access their vital online services.

Hackers use the same stolen secret to impersonate and access multiple apps, due to password re-use.



6: This problem compounds now that there are many companies acting on behalf of users to authenticate for them.

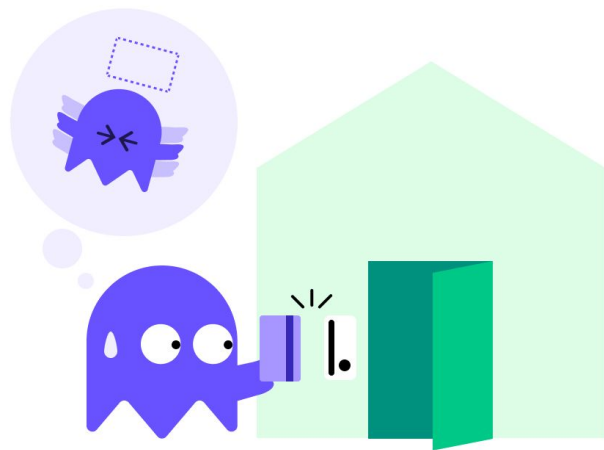
Users' identity is no longer in their own hands but controlled by a handful of large corporations.





What people tried before Magic

Zero-knowledge authentication



Key-Based Model

Instead of users thinking of secrets themselves, blockchain-based public-private key-pairs are randomly generated to access apps.

Pros:

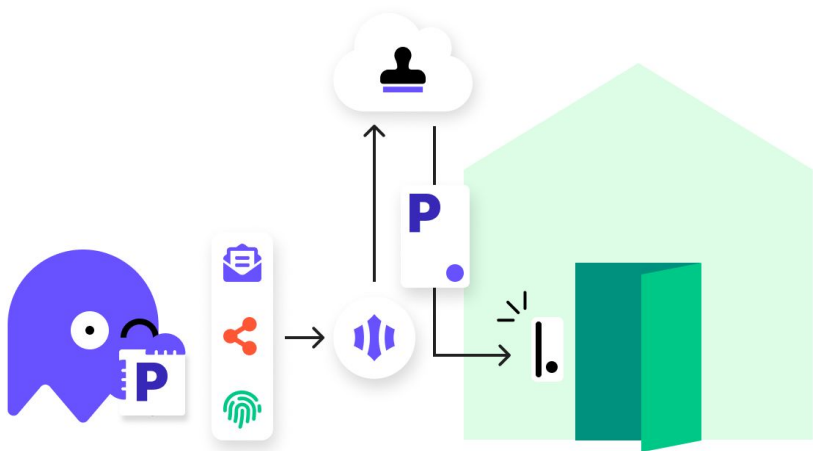
- Users have complete control of their identity
- Companies can't know users' secrets (more secure)
- Users can use the same secret to access apps - cleaner trust model where there are no identity silos by companies

Cons:

- Users are likely to lose their keys, which will lock them out for good and lose their online identity, or worse get them stolen
- The concept of using a key on the Internet is too unfamiliar for most mainstream users (bad UX)

How Magic improved the trust so far

Delegated Key Management



Delegated Model

Magic leverages large IaaS and secure user keys with HSMs, with technology that hides and protects user private keys from companies and *even* Magic.

Additional Pros:

- Magic provides familiar passwordless auth UX to users for them to retrieve their keys (better UX, can no longer lose keys)
- Magic doesn't store passwords and can't know users' keys / secrets (more security & trust)
- Native support for multiple blockchains

Additional Cons:

- Reliance on Single IaaS

What is **DID Token**?

Decentralized IDentity Token - DID Token

- DID Token created by the **Magic**, is adapted by prior tech like JWT and W3C's DID Protocol.
- It is encoded as a Base64 JSON string tuple representing [proof, claim]
- It leverages the **Ethereum** blockchain and *elliptic curve cryptography*
- To generate *verifiable proofs of identity* and *authorization*.
- These **proofs** are encoded in a *lightweight, digital signature*
- Which is shared between **client** and **server**
- to *manage permissions; protect routes* and *resources*, or *authenticate users*.

Decentralized IDentity - DID

The DID token is encoded as a Base64 JSON string tuple representing `[proof, claim]`:

- `proof`: A digital signature that proves the validity of the given `claim`.
- `claim`: Unsigned data the user asserts. This should equal the `proof` after Elliptic Curve recovery.

```
const claim = JSON.stringify({ ... }); // Data representing the user's access
const proof = sign(claim); // Sign data with Ethereum's `personal_sign` method
const DIDToken = btoa(JSON.stringify([proof, claim]));
```

Copy



Generating a DID Token (Pseudo-code)

```
// Construct the user's claim
const claim = JSON.stringify({
  iat: Math.floor(Date.now() / 1000),
  ext: Math.floor(Date.now() / 1000) + lifespan,
  iss: `did:ethr:${user_public_address}`,
  sub: subject,
  aud: audience,
  nbf: Math.floor(Date.now() / 1000),
  tid: uuid(),
});

// Sign the claim with the user's private key
// (this way the claim is verifiable and impossible to forge).
const proof = sign(claim);

// Encode the DIDToken so it can be transported over HTTP.
const DIDToken = btoa(JSON.stringify([proof, claim]));
```

Decentralized ID Token Specification

Key	Description
iat	Issued at timestamp (UTC in seconds).
exp	Expiration timestamp (UTC in seconds).
nbf	Not valid before timestamp (UTC in seconds).
iss	Issuer (the signer, the "user"). This field is represented as a Decentralized Identifier populated with the user's Ethereum public key.
sub	The "subject" of the request. This field is populated with the user's <i>Magic entity ID</i> . Note: this is separate from the user's Ethereum public key.
aud	Identifies the project space. This field is populated with the <i>application's Magic entity ID</i> .
add	An encrypted signature of arbitrary, serialized data. The usage of this field is up to the developer and use-case dependent. It's handy for validating information passed between client and server. <i>The raw data must already be known to the developer in order to recover the token!</i>
tid	Unique token identifier.



Generating a DID Token with Magic

```
import { Magic } from 'magic-sdk';  
  
const m = new Magic('API_KEY');  
  
// log in a user by their email  
  
try {  
  await m.auth.loginWithMagicLink({ email: 'hello@example.com' });  
  const DIDToken = await m.user.getIdToken({ lifespan? = 900 });  
} catch {  
  // Handle errors if required!  
}
```






Authentication & Authorization

https://dashboard.magic.link/app/ x +

dashboard.magic.link/app?cid=FqXz0LTtE7z7hSpGIPYO7c_Gv0DuPHIB... | Private (2)



 

First App ▾

-  Home
-  Users
-  Branding
-  Social Login
-  Settings

Install Magic

Grab your key & follow the Get Started docs to add Magic to your app:

API KEY	SECRET KEY
<code>pk_live_B909595EACA450...</code> 	Reveal Secret 

Created: Fri May 28 2021 [Roll Keys](#)

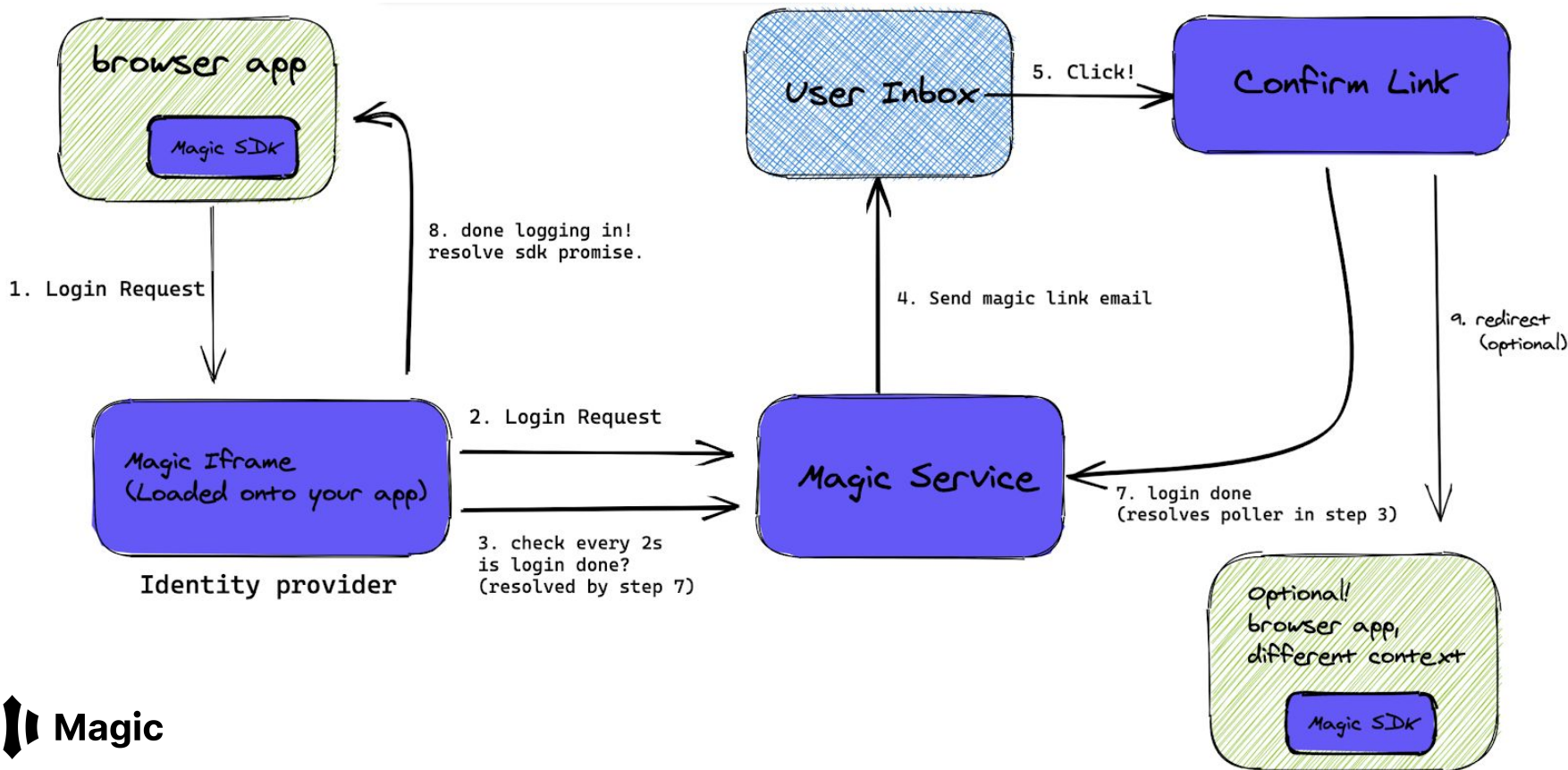
[Get Started](#)

Total funds ?

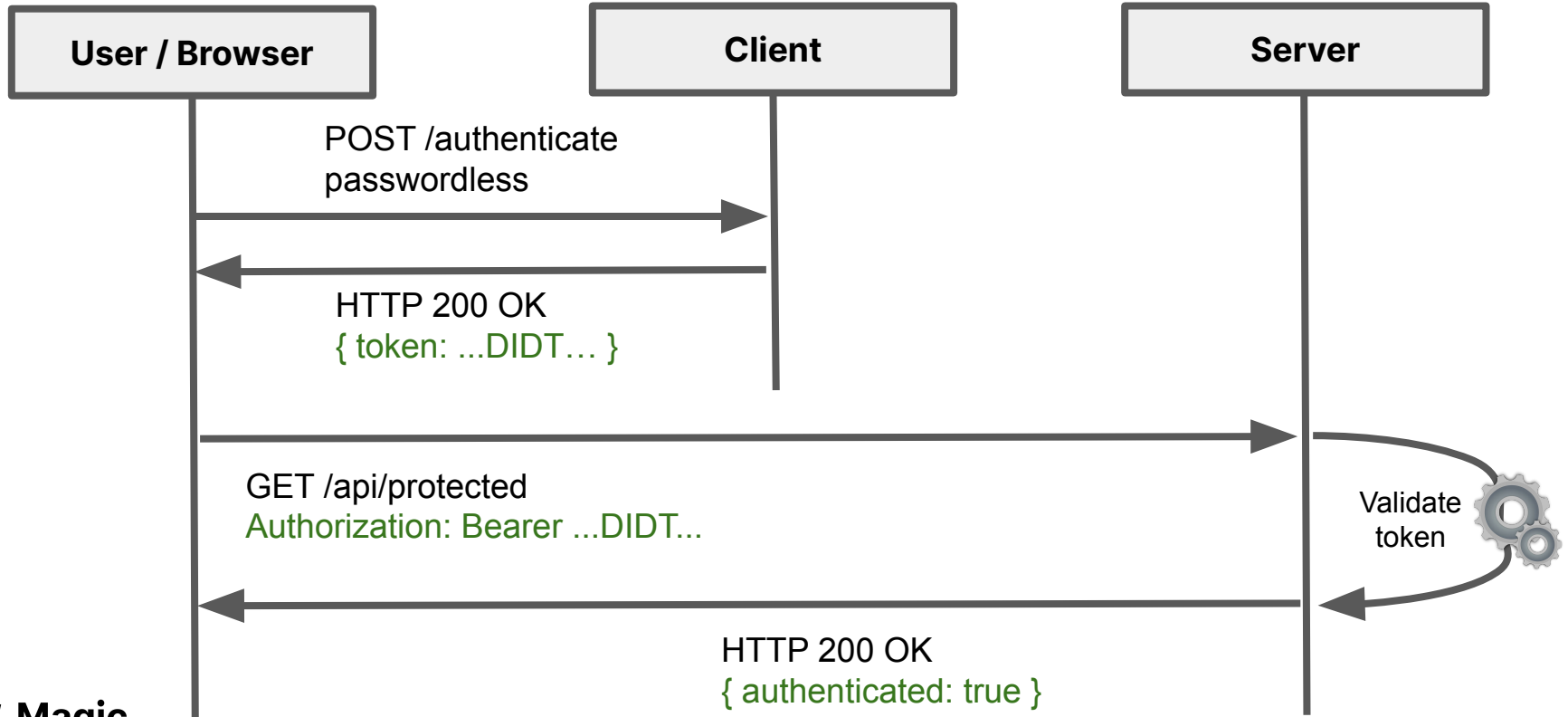


@mdsbzalam

Authentication (loginWithMagicLink):



Auth flow



Build **Go** API

Get started locally in seconds

Sound too good to be true? Take Magic for a spin. Bootstrap your project using a simple CLI tool that generates a fully working application with Magic auth built-in.

```
npx make-magic
```

Next.js - Frontend



```
npx make-magic --template next
```

Resources

Frontend: <https://github.com/shahbaz17/frontend-go-api>

Go Server: <https://github.com/shahbaz17/magic-go-api>

Magic Go Admin SDK: <https://docs.magic.link/admin-sdk/go/get-started>

Decentralized Identity: <https://docs.magic.link/decentralized-id>

DID: <https://w3c-ccg.github.io/did-primer>

DID Token: <https://docs.magic.link/decentralized-id#what-is-a-did-token>

Magic Docs: <https://docs.magic.link>

Magic Guides: <https://magic.link/guides>

Magic Community: <https://community.magic.link>

Connect with me

npx mdsbzzalam

Twitter: [@mdsbzzalam](https://twitter.com/mdsbzzalam)

LinkedIn: <https://www.linkedin.com/in/mdsbzzalam>

Youtube: <https://www.youtube.com/c/mdsbzzalam>

Github: <https://github.com/shahbaz17>

Dev.to: <https://dev.to/shahbaz17>

Email: shahbaz@magic.link

Website: mdsbzzalam.dev



@mdsbzzalam

Thank you

CONF42

GOLANG



@magic_labs