Magic: A Key-Based Authentication System For Self-Sovereign Identity

https://magic.link

Magic Labs, Inc.
March 23, 2020

Abstract

Managing user identities for authentication and authorization are serious responsibilities. Unauthorized access can lead to breaches of user identity, privacy and even financial losses. To manage this, performing penetration tests and audits, and staying on top of current security trends are *huge* endeavors and may require full-sized security teams to execute correctly. In an age of ever-increasing cybersecurity threats, more and more software companies are shying away from undertaking these high levels of responsibility and liability.

Worldwide spending on cybersecurity is projected to reach $133.7 billion in 2022 [1], and the average cost of a breach has skyrocketed to around $3.92 million as of 2019 [2]. Although Equifax was breached in 2017, the company is still paying off the $4 billion in total damages.

Magic provides a key-based identity solution built on top of the Decentralized Identity (DID) standard, where users' identities are self-sovereign by leveraging blockchain public-private key pairs. These key pairs are used to generate zero-knowledge proofs to authenticate users instead of having to rely on users providing passwords to Magic or any identity provider. By leveraging elliptic curve cryptography, these proofs can be used to securely communicate with developers' own resource servers. Since Magic's authentication protocol is based on key pairs provided by decentralized blockchain

networks, it is platform-independent and thus able to provide authentication service without having to rely on centralized identity providers.

Table of Contents

Introduction

Proper management of user credentials, passwords, secrets, and sessions require a tremendous amount of resources. Poor passwords actually account for 81% of all security breaches, since over 59% of people reuse their passwords *everywhere* [3]. This will cost companies up to $240,000 for every 1,000 records compromised [4] - increasing risk and liability for your company.

By typing your email in HaveIBeenPwned.com, you can realize that your sensitive data and passwords have likely been compromised in many high profile breaches impacting companies like Equifax, Dropbox, Adobe, Kickstarter, LinkedIn, Tumblr, and so many more.

Once a database of hashed passwords is stolen, hackers can direct immense distributed computing resources at that database of passwords, utilizing parallel GPUs or botnets with hundreds of thousands of nodes to attempt hundreds of billions of password combinations per second in hopes of recovering plaintext identifier+password pairs.

Once attackers discover a password that hashes to the same hash as the one stored in the database, they'll take that identifier+password pair and try it on other applications like bank accounts. In many cases, a salted+hashed password database can give up another valid identifier+password pair around every minute. This results in about half a million leaked passwords per year on its own — doubling every few years.

For companies using passwords as secrets to encrypt sensitive data, it's very much like symmetric encryption where the encryption key is a weak password determined by humans that can be easily cracked by brute-force.

In addition, over 50% of all support tickets are password-related (usually forgotten passwords). Each ticket costs companies around $70 - handling 10 tickets daily will cost an organization close to $128k annually [5]. A secure passwordless identity solution will significantly reduce a company's operation cost, risk, and liability.

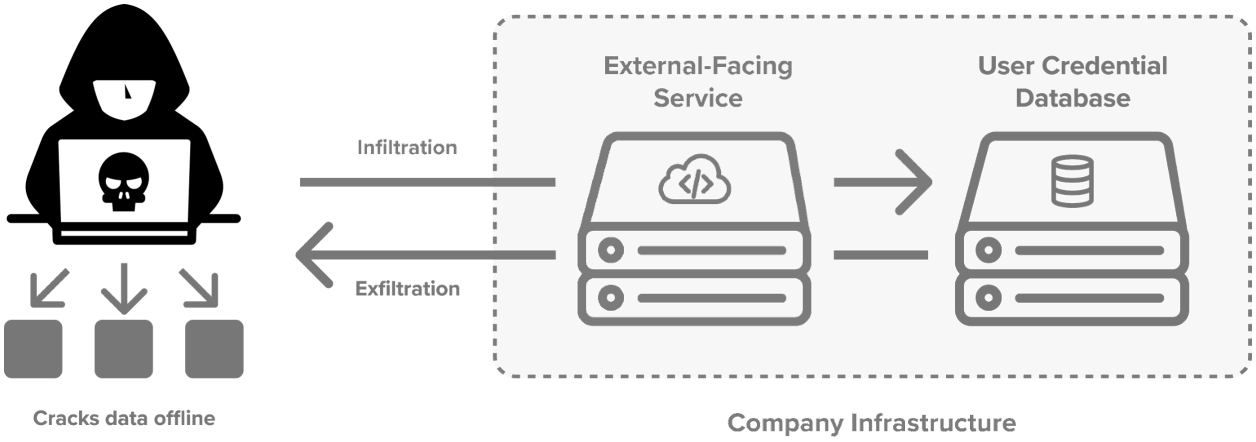
Architecture

Canonical Exfiltration Attacks

To protect against data breaches, one needs to understand how modern-day data-exfiltration attacks are executed:

1. **Recon** - Attacker researches, profiles, and tests the victim company's environment and people.
2. **Infiltrate** - Attacker breaks in or takes positions inside the organization.
3. **Discover** - Attacker leverages an internal position to learn more about the system.
4. **Capture** - Attacker takes control of valuable user data (including passwords and credentials).
5. **Exfiltrate** - Attacker moves encrypted data out, uses offline compute resources to crack them.
6. **Monetize** - Attacker sells or uses decrypted data to make money or gain an advantage.

What can be more catastrophic is that, since encrypted data can be cracked offline, sometimes the victim company won't even realize they are compromised until months later, if ever.



In order to prevent an attacker from cracking user password credentials and data offline, we have to (1) figure how users can authenticate into applications without passwords, and (2) ensure sensitive user data are "locked" by something that can't physically leave Magic's infrastructure.

Zero-Knowledge Authentication

Traditional user|password authentication is done by matching a secret string set and memorized by a user. To initiate authentication, the server presents the user with a “Challenge”, namely “what is your password?” The correct password is stored on the challenger (developer’s) resource server. The user’s answer to the challenge is sent via HTTP requests to the server, which then proceeds to match the provided password with the expected one.

One feature endemic to the blockchain's public|private key Elliptical Curve Cryptography is the *Zero Knowledge Proof*. A private key owner does not need to *reveal* his or her private key in order to prove that they own the key (and by corollary, that they are who they say they are). What the user can do instead is sign a piece of data with their private key, which generates a signature, and send it as proof. This signature is cryptographically secure, as it is computationally infeasible to derive private keys from signatures. The server runs a standard, stateless cryptographic recovery method on the signature which validates it as either authentic or not.

As applied to identity and authentication, this means that the challenger no longer has to send a secret to the server in order to validate his/her identity, and the server no longer has to store this class of sensitive information in order to validate users. Instead of storing passwords, the server only has to map users based on their *public* keys, which act as unique identifiers of private keys, and cannot themselves be used to generate signatures. This authentication model has been widely adopted by decentralized applications (dapps) and are battle-tested in the Ethereum blockchain ecosystem in forms of standardized data signature protocols such as the EIP-712 standard.

Decentralized Identity Token

One of the central benefits to Delegated Key Management is the promise of a transparent and secure mechanism through which users grant their consent. By signing data client-side with their private keys, users can generate a **stateless token representing a verifiable *identity***. In this section, we describe a protocol for encoding and decoding these messages using standardized Elliptic Curve cryptography: the **Decentralized Identity Token (DIDT)**.

DIDT is inspired by prior art like JSON Web Token (JWT) and W3C's Decentralized Identifiers (DID) protocol. The basic construction of a DIDT is JSON-based as well, represented as a Base64-encoded tuple with two elements:

- The Elliptic Curve signature of the token data, used to verify the token's integrity.
- The plaintext data encoded by the token to which the signature is recovered along with the signer's public address.

The plaintext data represents a *claim* object: a collection of assertions made by the signer at the time the signature is generated. These assertions include: Time of issuing signature (*iat*); expiry time of token (*ext*); issuer of token (*iss*); subject matter of token (*sub*); intended audience of token (*aud*), do

not use before time (*nbf*); a unique token id (*tid*); and an optional attachment field (*add*) containing a nested signature of miscellaneous data, signed with the same private key.

$C = \{\, iat\colon\ Current\ Time,\ \ldots,\ add\colon\ S(D)\ \}$ ***[*S: Cryptographic Signature, *D: Arbitrary Data]***

$P\ =\ S(C)$ ***[*S: Cryptographic Signature]***

$T\ =\ [P,\ C]$ ***[*T: DID Token as a base64 encoded string]***

The intended use of the DID Token is to provide a stateless, vendor-less proof of identity and authentication. Users leverage our key management system to generate the Token, and send it inside https requests to an app's resource server in order to initiate secure sessions and access restricted resources. With Decentralized Identity, developers no longer need to manage a user's private keys or passwords in order to verify that a user has given consent to access sensitive resources. Anyone can verify the integrity of a DID Token but only the owner of the private key can generate one.

One benefit of the DID Token mechanism over traditional JWT standards is the decentralization of secret keys used to generate authentication tokens. JWTs are issued via a centralized identity provider; if that centralized authority is compromised, it becomes possible for malicious parties to generate counterfeit tokens. DID Tokens are signed using private keys stored in a decentralized manner. The breach of a single private key does not compromise the entire platform. The only known way to break the integrity of the DID Token on a protocol level is to exploit a flaw in the mathematics of Elliptic Curve cryptography.

Delegated Key Management

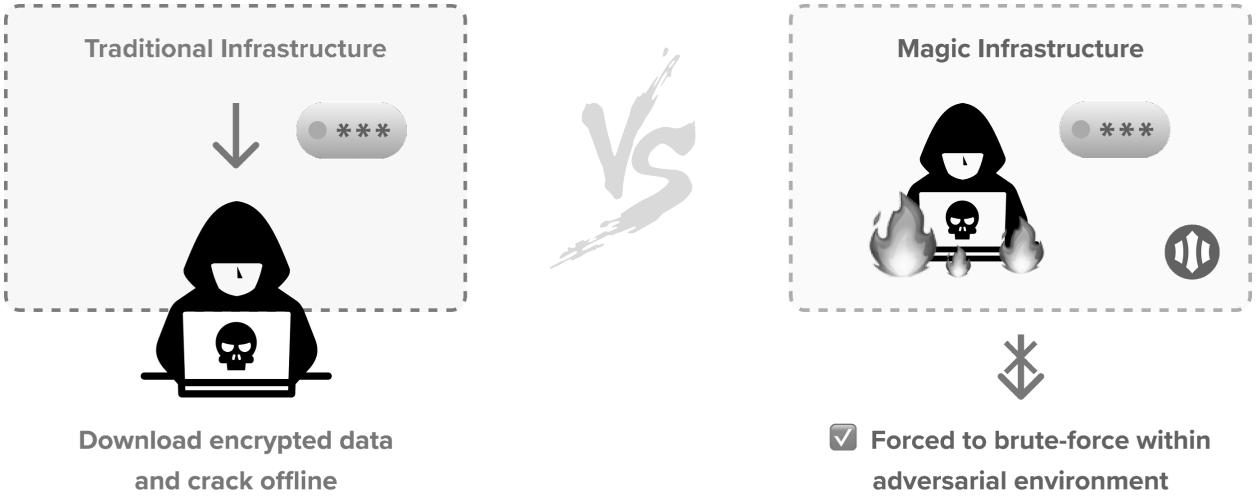
This public-private key approach means that the user's identity is effectively bound to their private keys. Therefore, it is critical to ensure that users' private keys are properly secured. Traditional databases are susceptible to data-exfiltration attacks, which has prompted the development of our own proprietary Delegated Key Management (pat. pend. USPTO 62/904689) System.

Hardware Security Modules

Magic's Delegated Key Management leverages Hardware Security Modules (HSMs) provided by Amazon Web Services' Key Management Service (AWS KMS). Dedicated user master keys generated using AES-256 with 384-bits of entropy are stored on the HSMs. The master keys never leave the hardware as they are meant to be locked inside and unable to be exported. All encryption and decryption operations happen inside the hardware modules themselves. HSMs are a lot like popular

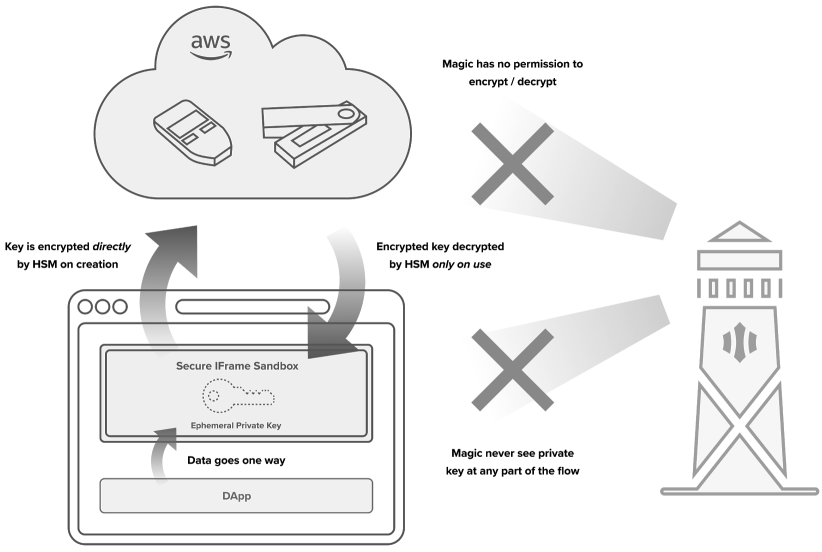
FIDO devices like YubiKeys or hardware-based wallets for cryptocurrency storage such as Trezor or Ledger, but instead of traditional harddrive storage, they sit in the cloud, secured by AWS's data centers.

Users' private keys are encrypted by these hardware-based user master keys, which means that attackers need to gain access to the hardware to be able to retrieve the keys, and are forced to stay *within* Magic's adversarial infrastructure - which is capable to detect, impede, and monitor attacker's progress to prevent and mitigate any damages.



Non-Custodial Trust Optimization

Our Key Management architecture *delegates* critical encryption and decryption operations to a trusted AWS KMS and AWS Cognito - completely bypassing the Magic backend. This lets us stay non-custodial (standard for trust amongst crypto communities) as user private keys are never seen by our systems.



Client to AWS Data Flow

For users to access their HSMs, they authenticate with the Magic auth relayer. Upon successful authentication, users receive a time-bound access token, which is traded for scoped credentials, and in turn, allows the client to interact with AWS directly. The scoped credentials exclusively enable *users* to call the AWS to access their master keys stored on the HSM for encryption and decryption. This mechanism bypasses the Magic backend entirely, and Magic cannot forge nor intercept the scoped credentials because they are generated by the operations exclusively between the clients and AWS. Both access tokens and scoped credentials are created dynamically by AWS, with audit logs, and with TTLs enabled.

When a user signs up, a public-private key-pair is generated on the client-side for the user inside a javascript browser iframe, inaccessible by the integrating app. Keys are generated with cryptographically secure pseudo-random 256-bit entropy. Once generated, the key is encrypted by the Delegated Key Management System, where no parts of the raw private key are seen by the Magic infrastructure. After encryption, the *encrypted* key is stored on the client-side iframe as well as uploaded to the Magic auth relayer for redundancy, in case the encrypted key on the client-side is accidentally removed.

When users authenticate again, the encrypted private key is downloaded to the client. Users can then decrypt the encrypted key directly with AWS KMS via Cognito.

Phishing Protection

Phishing protection is under-taken in two different aspects.

Passwordless Authentication

Magic users do not set/memorize passwords for authentication. Instead, a Magic Link email is sent to the user's mailbox. The Delegated Key Management credentials are loaded onto the client iframe after the Magic Link is clicked. This way, Magic eliminates the case where users can be phished for account passwords. Moreover, if a magic link email is lost or stolen (or even somehow compromised in transit), a user's account remains safe. The token included in the magic link email is only privileged to verify a login request *from the device and/or browsing context that initiated the request.* An attacker would require physical access to the user's device *and* unencrypted email inbox to be malicious.

Domain Whitelisting

A motivated attacker could create an identical replica of your application. For this case, Magic offers the ability to whitelist **specific domains for their public API keys on the Magic Dashboard** so that illegitimate applications cannot forge requests through the Magic SDKs.

CSRF and XSS

CSRF and XSS are all the common attack vectors for web applications. Magic regularly audits 3rd-party scripts and enforces strict CSP headers to mitigate XSS and untrusted data ingestion exploits. CSRF tokens are used to ensure requests' authenticity, which prevents users' requests from being forged by malicious attackers. We adopt industry-best engineering practices building our products to make sure we are not vulnerable to any untrusted data injection. We are also rigorously implementing security controls and monitoring to protect the end-users further.

E2E TLS Encryption

Transport Layer Security (TLS) is the standard protocol for encrypting data on the Internet. Web services provided over the Internet without TLS should be strongly discouraged. At Magic, all service

communication is forced to be on TLS. All user data transported between Magic servers and user browser are encrypted end-to-end. This also applies to the 3rd-party services that we made available to the end-users. The communication between our servers and the internal application secret storage, Hashicorp Vault, is also encrypted by the TLS. In the unlikely event of Magic servers being infiltrated, none of the sensitive data transported on the internal network is in plaintext.

By implementing TLS, it prevents man-in-the-middle attacks. Users using services provided by Magic are protected end-to-end.

Network Setup

Magic deploys services in virtual private clouds (VPC). Each environment (dev, stage, and prod) has a dedicated VPC, and no communication is allowed between VPCs except with the VPN's VPC. Within a VPC, we split up our network into public subnets and private subnets. Public subnets are the only network with Internet access. Services deployed in public subnets are accessible to the world, whereas the ones in the private subnets aren't. This allows us to deploy core business logic in the private network and protect those services from the Internet.

Intrusion Detection Setup

Magic deploys intrusion detection systems in its cloud environment. We analyze all of our VPC logs, DNS logs and service logs to detect threats and any unauthorized access. We continuously monitor the traffic and take action when needed.

Data Encryption at Rest

Magic encrypts all of its databases, snapshots, automated backups and replicas with the industry-standard AES-256 encryption algorithm. The encryption and decryption are done automatically when data is written and read from the data storage.

Keypair Equivalency Classes

Within the blockchain ecosystem there are equivalency classes with a keypair such as hierarchical deterministic wallets. In all such cases these equivalent classes are secured in the same way as a keypair. For example, in the case of an HD wallet, the mnemonic is secured in our DKMS system. A derived key from an HD Wallet will also be secured in our DKMS system.

Security Audit and Compliance

SOC 2 Type 1

Magic's infrastructure has been thoroughly audited by **NCC Group** and **A-LIGN** and is **SOC 2 Type 1** compliant (SOC 2 Type 2 in progress). All the controls and procedures required by the compliance have been implemented at Magic.

Security and trust are the top priority. They're not just architecture, but also intensive process and diligence around privacy, confidentiality, risk mitigation, and business continuity. Magic is also insured for cybercrime damage and loss.

What's covered in our SOC 2 Information Security Management System (ISMS)

Technology	**Process**	**Business**
Encryption and Key Management	Privacy and Security Training	Business Continuity
Incident Response	Secure Information Handling	Risk Management
Network & Remote Access	Vulnerability Management	Information Privacy & Security Governance

Identity and Access Management	Asset Management	Physical and Environmental Security
Device Security	Configuration and Change Management	Vendor Management
Email and Messaging	Data Classification	Human Resources Information Security
Logging	Data Retention and Disposal	Technology Acceptable Use

Conclusion

The first obstacle to a more authentic Internet is *identity*, before anything else can be possible. Identity has been predominantly gated by centralized identity providers, tech oligopolies such as Facebook and Google. Developers are completely dependent on these providers for user access, which directly impacts the livelihood of their businesses. Rolling authentication and security from scratch is extremely challenging to keep secure, privacy-compliant, scalable, and reliable - leading to vast amounts of user data and credential breaches.

Magic solves this by "decentralizing" identity. Instead of centralized identity providers signing auth tokens, *users* are the ones signing auth tokens with their own private keys - enabling self-sovereign identity. This uses blockchain key pairs, elliptic curve cryptography, and W3C's decentralized identity protocol.

Our delegated key management solution represents a pragmatic way of onboarding users into this new paradigm of cryptographic ID by decentralizing trust. Users do not need to store private keys themselves in order to interact with the authentication system, providing an approachable middle ground that will serve as the foundation for an eventual, fully decentralized, authentic internet. Solving

identity with blockchain is the first step that will enable new business models emphasising trust, distributed serverless compute, storage, networking, and financial services.

References

[1] Susan Moore, Emma Keen, “Gartner Forecasts Worldwide Information Security Spending to Exceed $124 Billion in 2019”, https://www.gartner.com/en/newsroom/press-releases/2018-08-15-gartner-forecasts-worldwide-information-security-spending-to-exceed-124-billion-in-2019, Aug 2018

[2] Larry Ponemon, “What’s New in the 2019 Cost of a Data Breach Report”, https://securityintelligence.com/posts/whats-new-in-the-2019-cost-of-a-data-breach-report/, Jul 2019

[3] TraceSecurity, “81% of Company Data Breaches Due to Poor Passwords”, https://www.tracesecurity.com/blog/articles/81-of-company-data-breaches-due-to-poor-passwords, Aug 2018

[4] Chris Brook, “What’s the Cost of a Data Breach in 2019?”, https://digitalguardian.com/blog/whats-cost-data-breach-2019, Jul 2019

[5] Daniel, Lu, “How Much Are Password Resets Costing Your Company?”, https://www.okta.com/blog/2019/08/how-much-are-password-resets-costing-your-company/, Aug 2019