30 SEPTEMBER 2018 / #API

Best practices for building secure API Keys

by Ramesh Lingappa

We all know how valuable APIs are. They're the gateway to solutions faster.

You might have built or are thinking of building APIs for other developers to use. An API needs some form of authentication to provide authorised access to the data it returns.

There are several authentication standards available today such as API Keys, OAuth, JWT, etc.

In this article, we'll look at how to correctly manage API Keys to access APIs.

So Why API Keys?

API Keys are simple to use, they're short, static, and don't expire unless revoked. They provide an easy way for multiple services to communicate.

If you provide an API for your clients to consume, it's essential for you to build it in the right way.

Let's get started, and I'll show you how to build API Keys the right way.

API Key Generation

Since the API key itself is an identity by which to identify the application or the user, it needs to be unique, random and non-guessable. API keys that are generated must also use Alphanumeric and special characters. An example of such an API key is `zaCELgL.0imfnc8mVLWwsAawjYr4Rx-Af50DDqtlx`.

Secure API Key Storage

Since the API key provides direct access to data, it's pretty much access to the same data.

Think about it. The reason we need to store API keys is to make sure that the API key in the request is valid and issued by us (just like a password).

We don't need to know the raw API key, but just need to validate that the key is correct. So instead of storing the key in plain text (bad) or encrypting it, we should store it as a hashed value within our database.

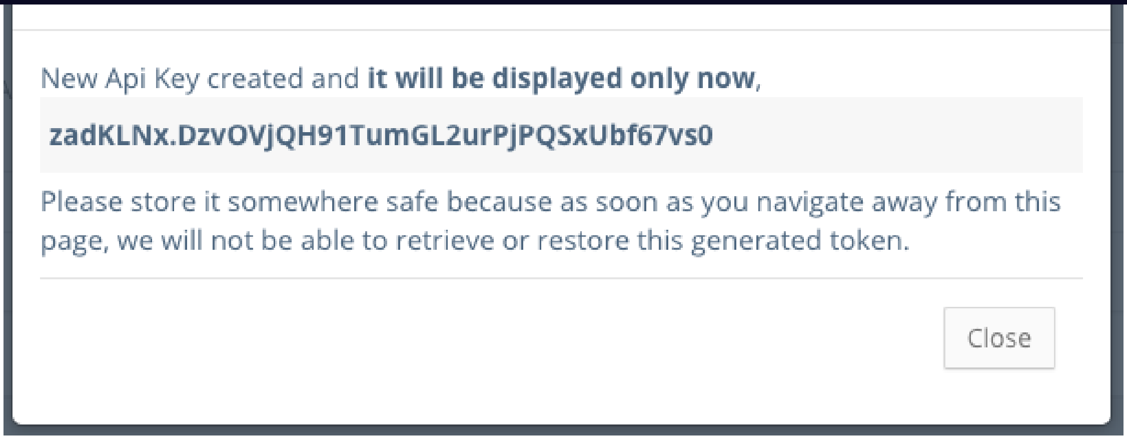
A hashed value means that even if someone gains unauthorised access to our database, no API keys are leaked and it's all safe. The end user would send the raw API key in each API request, and we can validate it by hashing the API key in the request and compare the hashed key with the hash stored within our database. Here is a rough implementation of it in Java:

In the code above, the primary key will be a combination of the prefix and the hash of the API key `{prefix}.{hash_of_whole_api_key}`.

But hold on, there is more. Storing a hashed value brings specific usability problems. Let's address those now.

Presenting the API Key to users

Since we don't store the original API key, we can show it only once to the user, at the time of creation. So be sure to alert users that it cannot be retrieved again, and they need to generate a new token if they forget to copy the API key and store it safely. You can do something like this:



New Api Key created and **it will be displayed only now**,

zadKLNx.DzvOVjQH91TumGL2urPjPQSxUbf67vs0

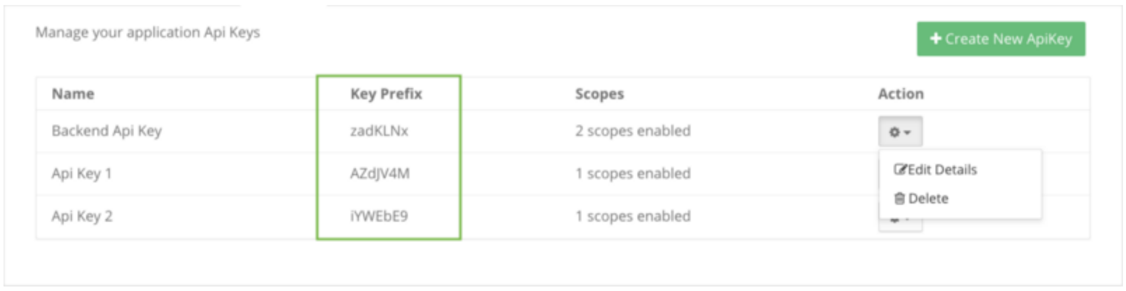
Please store it somewhere safe because as soon as you navigate away from this page, we will not be able to retrieve or restore this generated token.

Displaying generated API Key with an alert message

How users can identify a generated API Key later

Another problem is how users identify the right API key in your console if they need to edit or revoke it. This can be solved by adding a prefix to the API key. Notice in the picture above **the first 7 characters (that's our prefix),** separated by the dot.

Now you can store this prefix in the database and display it in the console so users are able to quickly identify the right API key entry, like this:



Manage your application Api Keys

Name	Key Prefix	Scopes	Action
Backend Api Key	zadKLNx	2 scopes enabled	⚙
Api Key 1	AZdJV4M	1 scopes enabled	
Api Key 2	iYWEbE9	1 scopes enabled	

Don’t give the API Key all the power

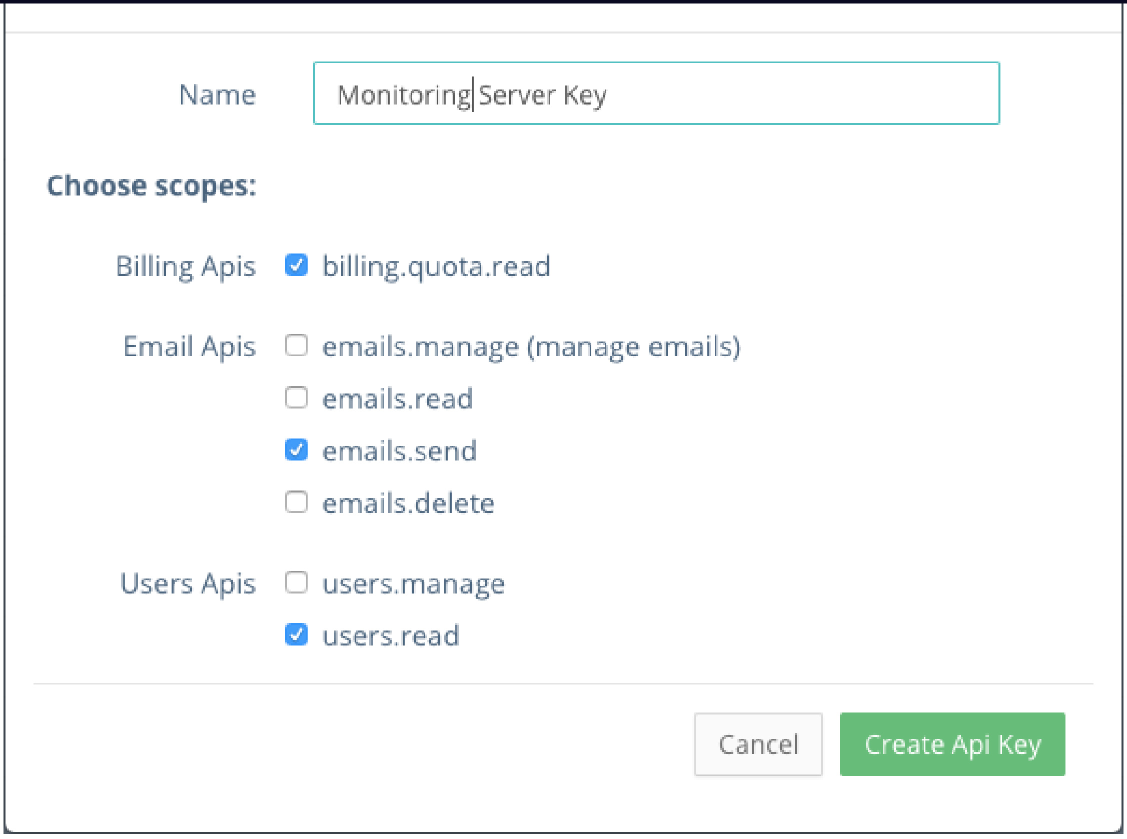
One common mistake that API key providers make is providing **one key to access everything**, since it’s easy to manage. Don’t do that. Assume that a user just needs to read an email, and generates an API key. But that key now has full access to other services, including deleting records in the database.

The **right** approach is to allow the end users to properly restrict API Key access and choose specific actions that an API key can carry out. This can be done by providing **scopes**, where each scope represents a specific permission.

For example,

- if you need an API key to just send emails, you can generate an API key with the scope as **“email.send”**
- if the end user has multiple servers and each carries out a specific action, then a separate API key can be generated with a specific scope.

So while creating the API key, allow users to select what access that API key should have, as in the image below.

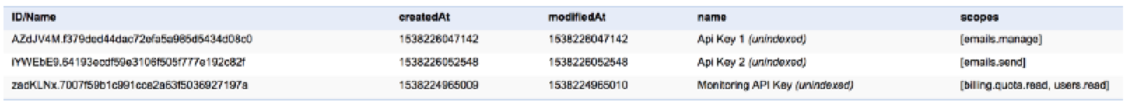


Name	Monitoring Server Key

Choose scopes:

Billing Apis	✓ billing.quota.read
Email Apis	☐ emails.manage (manage emails)
	☐ emails.read
	✓ emails.send
	☐ emails.delete
Users Apis	☐ users.manage
	✓ users.read

This way users can generate multiple API keys, each with specific rules of access for better security. And when an API request is received, you can check if the API Key has the right scope to access that API. Now the database looks something like this:



ID/Name	createdAt	modifiedAt	name	scopes
AZdJV4M.f379ded44dac72efa5e985d5434d08c0	1538226047142	1538226047142	Api Key 1 *(unindexed)*	[emails.manage]
lYWEbE9.64193ecdf59e3106f505f777e192c82f	1538226052548	1538226052548	Api Key 2 *(unindexed)*	[emails.send]
zadKLNx.7007f59b1c991cce2a63f5036927197a	1538224965009	1538224965010	Monitoring API Key *(unindexed)*	[billing.quota.read, users.read]

API Key database entity

Rate limiting API keys

Yes, you might already know it, but it is important to rate limit requests made with specific API Keys to ensure no bad actor can take down your API servers or cause performance issues that affect your other customers. Having a proper rate limiting and monitoring solution keeps the API service healthy.

API keys, when built right, are still a great way to communicate with another server. As we reviewed in this article, following certain practices offers benefits to both API consumers and API providers. Hope this helps you.

Happy Securing your APIs!

If this article was helpful, tweet it or share it.

Countinue reading about

API

Learn how to use APIs with React by building a Hacker News API application

An Introduction to JavaScript's queueMicrotask

A Python project in 30 lines of code: how to set up an SMS notification when your favorite Twitcher is streaming

See all 67 posts →

freeCodeCamp(🔥) Menu

#PROGRAMMING

How to stay motivated when learning to code (10 actionable tips!)

JESSICA CHAN A YEAR AGO

How to create Angular 6 Custom Elements and Web Components

A YEAR AGO

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have freeCodeCamp(🔥) Menu

Our Nonprofit

About
Donate
Shop
Alumni Network
Open Source
Support
Sponsors
Academic Honesty
Code of Conduct
Privacy Policy
Terms of Service
Copyright Policy

Best Tutorials

Python Tutorial
Git Tutorial
Linux Tutorial
JavaScript Tutorial
React Tutorial
HTML Tutorial
CSS Tutorial
SQL Tutorial
Java Tutorial
Angular Tutorial
WordPress Tutorial
Bootstrap Tutorial

Best Examples

Python Example
JavaScript Example
React Example
Linux Example
HTML Example
CSS Example
SQL Example
Java Example
Angular Example
jQuery Example
Bootstrap Example
PHP Example

Trending Reference

2019 Web Developer Roadmap
Linux Command Line Guide
Git Reset and Git Revert
Git Merge and Git Rebase
JavaScript Array Map
JavaScript Array Reduce
JavaScript Date
JavaScript String Split
CSS Flexbox Guide
CSS Grid Guide
Create a Linux Sudo User
How to Set Up SSH Keys