

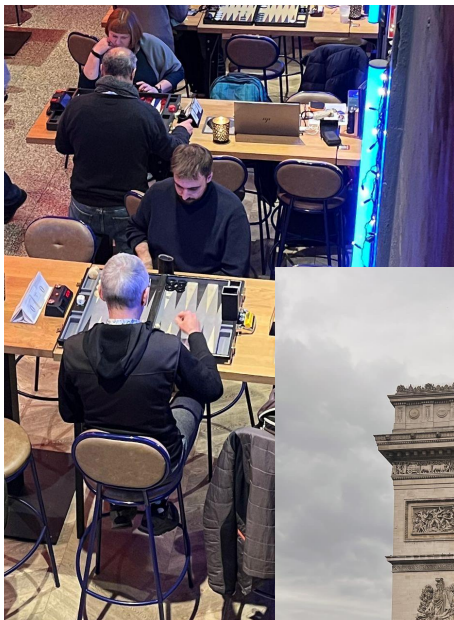
An aerial photograph of a long, multi-lane bridge spanning a large body of water. The bridge features a series of tall, thin piers supporting its structure. At the far end of the bridge, there is a small, star-shaped landmass with a road and some vegetation. The water is a deep blue, and the sky is a lighter blue with some clouds.

Keeping secrets secret in Django

Efe Öge

Hey, I'm Efe Öge

- Pronounced as “Eh-feh”
- Originally from **Türkiye**, based in **Amsterdam**
- Software Engineer @ Nebius
- Founder of yesglot.com  
- Writes at **efe.me** 
- DjangoNL & Mentor at Django Girls Amsterdam
(previously PyCon Turkey, Python Istanbul)



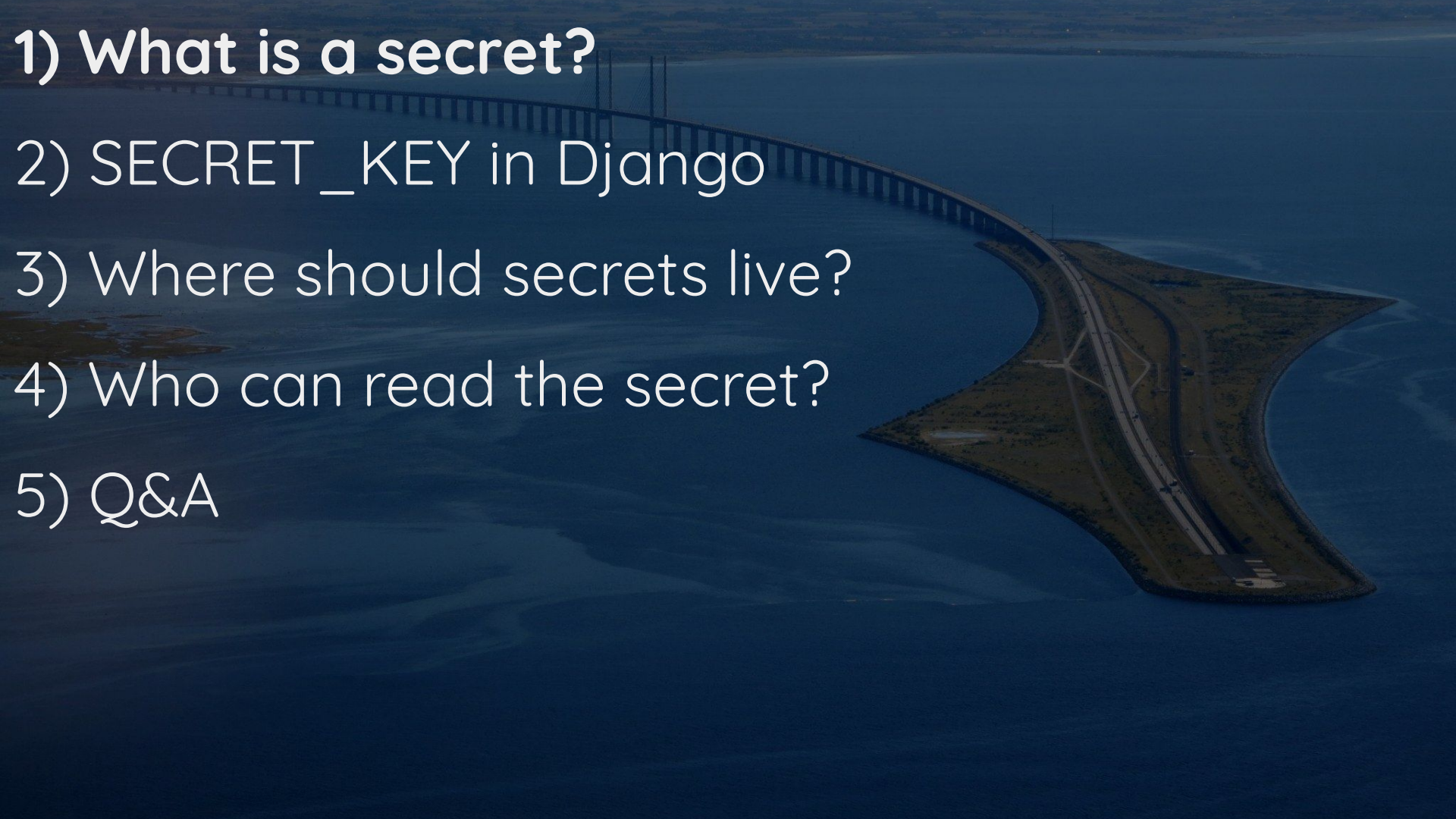
1) What is a secret?

2) SECRET_KEY in Django

3) Where should secrets live?

4) Who can read the secret?

5) Q&A



What is a secret? 🙊

What is a secret?

noun. a value whose disclosure **grants a capability** or **weakens a security guarantee**

- Database password
- Stripe secret API key
- GitHub access token
- OAuth client secret
- AWS secret key
- Django's **settings.SECRET_KEY**

What is not a secret?

- **Configuration** controls behaviour.
 - API base URL
 - ALLOWED_HOSTS
 - TIME_ZONE
- **Sensitive info** should not be disclosed.
 - Customer data in the database
 - Internal logs
 - Financial data
- **Public values** are designed to be shared.
 - OAuth client ID
 - SSH public key
 - Stripe public key

Secret = disclosure grants a capability

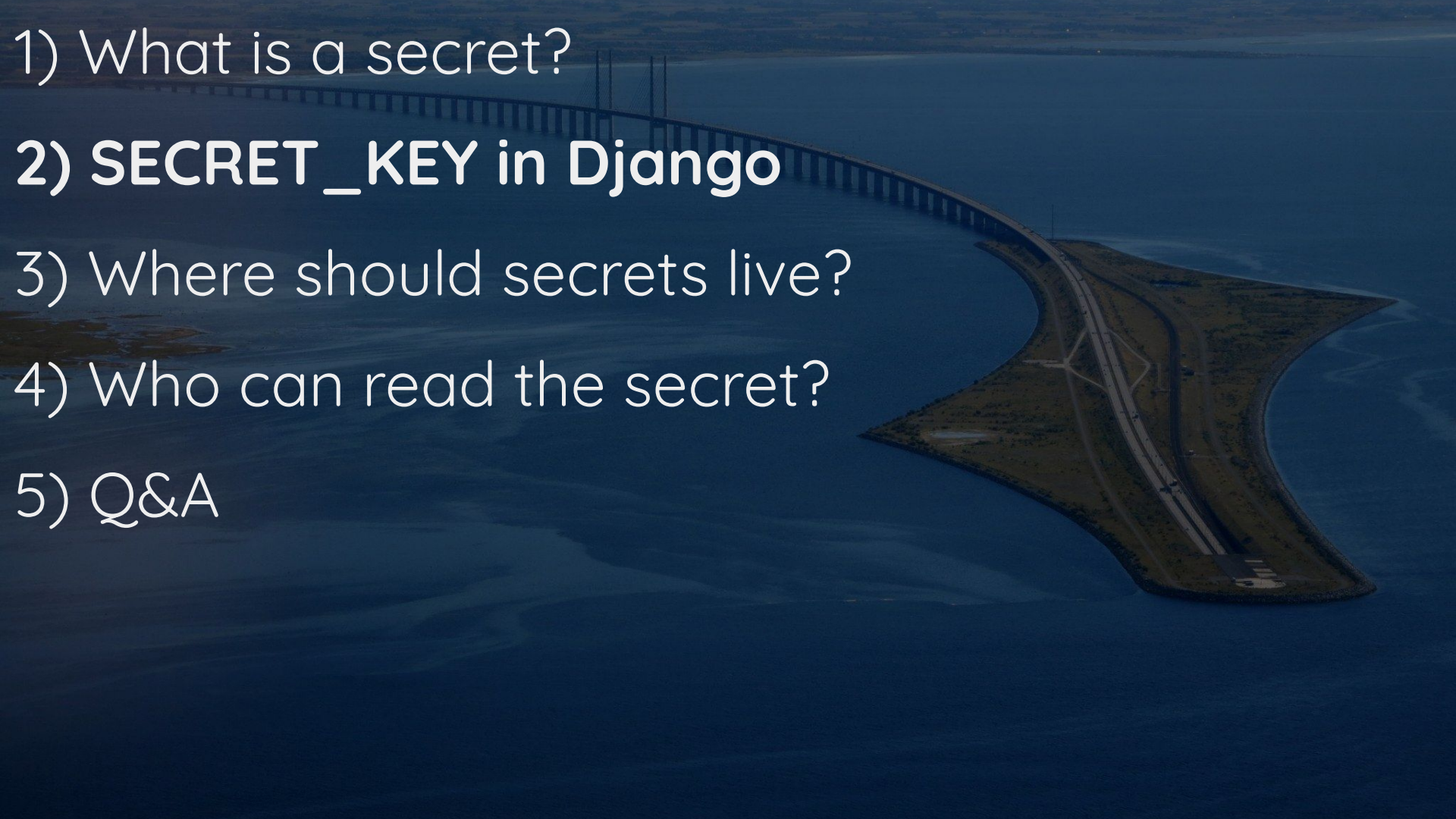
1) What is a secret?

2) **SECRET_KEY** in Django

3) Where should secrets live?

4) Who can read the secret?

5) Q&A



SECRET_KEY in Django



```
1 # settings.py
2
3 SECRET_KEY = "django-insecure-..."
```

but why?

Where is SECRET_KEY used?

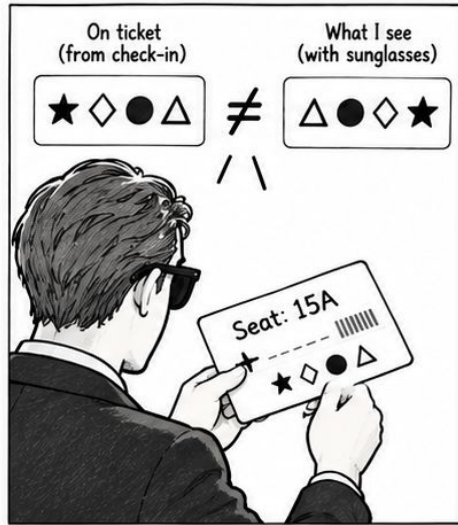
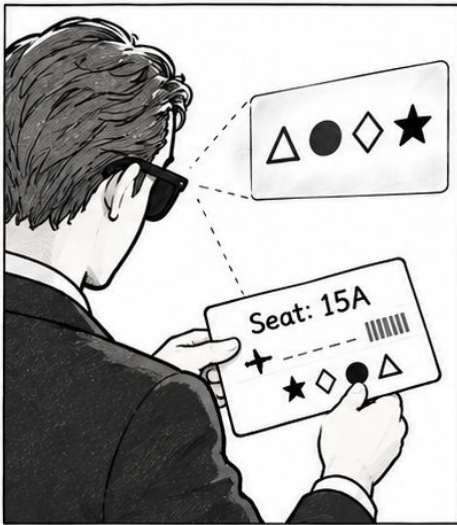
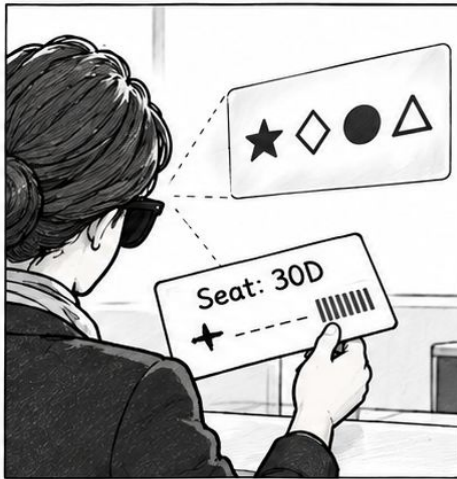
mainly used for

cryptographic signing.

The problem **cryptographic signing** solves

- We receive **a message**.
- We need to know: **is it original?**
- We can't **ask the sender every time**.
- What if the sender and receiver share **something** beforehand to check authenticity?





The problem **cryptographic signing** solves

Analogy	Technical Term
Boarding pass	Value / message
Special sunglasses	HMAC with SECRET_KEY
Figures seen on sunglasses	Signature*
Looking again with same glasses	Verification
Changed seat number	Tampered value

From Analogy to Django



```
1 from django.core.signing import Signer
2
3 signer = Signer()
4
5 signed = signer.sign("Seat: 30D")
6
7 print(signed) # "Seat: 30D:7f91a2..."
8
9 signer.unsign('Seat: 30D:7f91a2...') # 'Seat: 30D'
```

From Analogy to Django



```
1 from django.core.signing import Signer
2
3 signer = Signer()
4
5 signed = signer.sign("Seat: 30D") # "Seat: 30D:7f91a2..."
6 tampered = signed.replace("30D", "15A") # "Seat: 15A:7f91a2..."
7
8 signer.unsign(tampered)
9
10 Traceback (most recent call last):
11     ..
12     raise BadSignature('Signature "%s" does not match' % sig)
13 django.core.signing.BadSignature: Signature "q6Bq0Wb-
    dwFqgdGatsBijqBG9J_4ZgBuBUqvCz5jLHA" does not match
```

Where is SECRET_KEY?



```
1 # Django's source code: django.core.signing
2
3 class Signer:
4     def __init__(self, *, key=None, ...):
5         self.key = key or settings.SECRET_KEY
6
7     def signature(self, value, key=None):
8         key = key or self.key
9         return base64_hmac(
10             self.salt + "signer",
11             value,
12             key,
13             algorithm=self.algorithm,
14         )
15
16     def sign(self, value):
17         return f"{value}{self.sep}{self.signature(value)}"
```

Django's cryptographic signing APIs

API	Input	Adds timestamp?
<code>Signer.sign()</code>	string	No
<code>Signer.sign_object()</code>	Python object → JSON	No
<code>TimestampSigner.sign()</code>	String	Yes
<code>signing.dumps()</code>	Python object → JSON	Yes

Applications of cryptographic signing in Django

Application	Pattern
Signed-cookie session	Client stores data, server verifies integrity
Password-reset token	Time-sensitive proof
Session auth hash	Bind session validity to user state
Cookie-based messages	Detect modification of messages stored in the browser

Store less, verify more.

Quiz: How to generate a secure Django **SECRET_KEY**?

- a) Close my eyes and press random keys for 30 seconds
- b) Hash the current timestamp
- c) Use an online Django secret-key generator
- d) `django.core.management.utils.get_random_secret_key`

Quiz: How to generate a secure Django **SECRET_KEY**?

- a) Close my eyes and press random keys for 30 seconds
- b) Hash the current timestamp
- c) Use an online Django secret-key generator
- d) **`django.core.management.utils.get_random_secret_key`** ✓

Quiz: What will happen if you change SECRET_KEY

- a) Nothing. Thanks to SECRET_KEY, Django doesn't store any secrets
- b) Old signed values keep working until they expire
- c) Django automatically re-signs existing values with the new SECRET_KEY
- d) Old signatures can no longer be verified

Quiz: What will happen if you change SECRET_KEY

- a) Nothing — thanks to SECRET_KEY, Django doesn't store any secrets
- b) Old signed values keep working until they expire
- c) Django automatically re-signs existing values with the new SECRET_KEY
- d) Old signatures can no longer be verified ✓**

What will happen if you change SECRET_KEY

- all users are logged out
- existing sessions are invalidated
- existing messages may be lost
(*django.contrib.messages*)
- password-reset links are invalidated.
- and wherever you use signing.

But, can there be any other way?

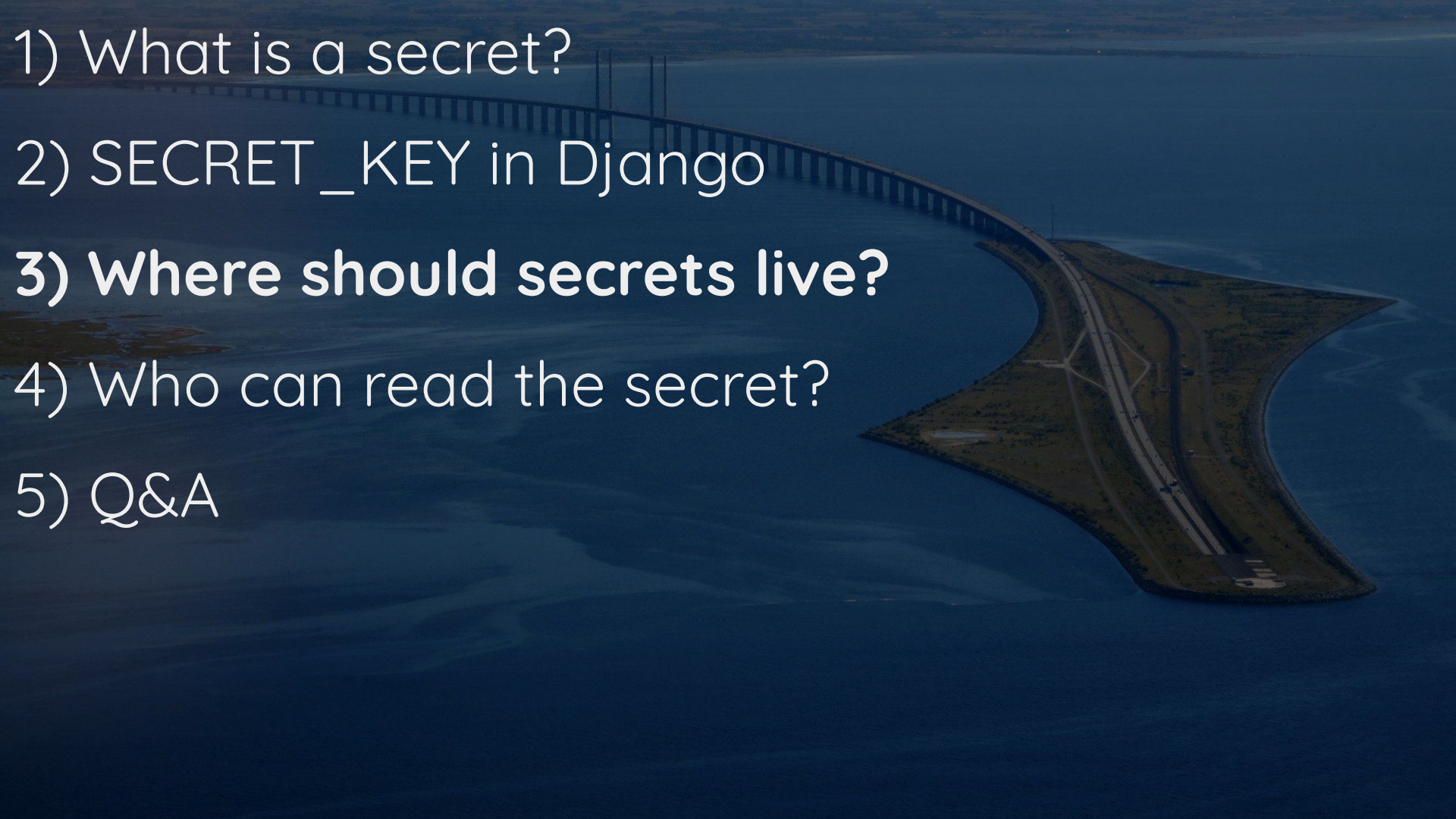
Django has solutions
for the problems

settings.SECRET_KEY_FALLBACKS



```
1 # settings.py
2
3 SECRET_KEY = "new-key"
4
5 SECRET_KEY_FALLBACKS = [
6     "old-key",
7 ]
```

Rotate SECRET_KEY gracefully.

- 
- An aerial photograph of a long, multi-lane bridge spanning a large body of water. The bridge has a series of tall, thin support pillars. At the end of the bridge, there is a small, triangular island with a road and some vegetation. The water is a deep blue, and the sky is a lighter blue.
- 1) What is a secret?
 - 2) SECRET_KEY in Django
 - 3) Where should secrets live?**
 - 4) Who can read the secret?
 - 5) Q&A

The convenient place



```
1 # settings.py
2
3 SECRET_KEY = "yjwt=w=h8w..."
4
5 DATABASES = {
6     "default": {
7         "ENGINE": "django.db.backends.postgresql",
8         "NAME": "myapp",
9         "USER": "myapp",
10        "PASSWORD": "super-secret-db-password",
11        "HOST": "db.example.com",
12        "PORT": "5432",
13    }
14 }
15
16 STRIPE_SECRET_KEY = "sk_live_..."
```

Move secrets out of the code



```
1 # settings.py
2
3 SECRET_KEY = os.environ["SECRET_KEY"]
4
5 DATABASES = {
6     "default": {
7         ...
8         "PASSWORD": os.environ["DATABASE_PASSWORD"],
9     }
10 }
11
12 STRIPE_SECRET_KEY = os.environ["STRIPE_SECRET_KEY"]
```

One secret becomes many

- SECRET_KEY
- DATABASE_PASSWORD
- STRIPE_SECRET_KEY
- AWS_SECRET_ACCESS_KEY
- SENDGRID_API_KEY
- SENTRY_TOKEN
- OAUTH_CLIENT_SECRET ...

One set of secrets, many environments

- development
- ci/cd
- staging
- production

Now every secret has to be provided
wherever the application runs.

Secrets multiply

Secret	Dev	CI/CD	Staging	Production
SECRET_KEY	✓	✓	✓	✓
DATABASE_PASSWORD	✓	✓	✓	✓
STRIPE_SECRET_KEY	✓	✓	✓	✓
SENDGRID_API_KEY	✓	✓	✓	✓
...	✓	✓	✓	✓

A missing one?

Secret	Dev	CI/CD	Staging	Production
SECRET_KEY	✓	✓	✓	✓
DATABASE_PASSWORD	✓	✓	✓	✓
STRIPE_SECRET_KEY	✓	✓	✓	✗
SENDGRID_API_KEY	✓	✓	✓	✓
...	✓	✓	✓	✓

What if we only had to provision one?

- 1) Create one `ROOT_ENCRYPTION_KEY` per environment.
- 2) Encrypt each secret with that environment's root key.
- 3) Commit the encrypted values to the codebase.
- 4) Provision only the root key to each environment.

main 1 Branch 5 Tags

Go to file

<> Code

efe uv.lock update ✓

7865994 · 6 months ago 19 Commits

.github/workflows	feat: implement publish.yml for auto release	7 months ago
django_root_secret	feat: get secret in encrypt_secret management command ...	7 months ago
.gitignore	chore: add .gitignore	7 months ago
LICENSE	Initial commit	7 months ago
README.md	feat: get secret in encrypt_secret management command ...	7 months ago
pyproject.toml	fix: Change the required Python version from >= 3.13 to >...	6 months ago
uv.lock	uv.lock update	6 months ago

[README](#) [MIT license](#)

django-root-secret

Tests passing

django-root-secret is a Django package for managing one root encryption key per environment and decrypting encrypted literals at runtime.

About

Django package for managing one root encryption key and decrypting encrypted secrets at runtime.

pypi.org/project/django-root-secret/

django

encryption

python

secrets

[Readme](#)[MIT license](#)[Activity](#)

☆ 20 stars

👁 0 watching

🍴 2 forks

Report repository

Releases

🏷 5 tags

Contributors 2



efe Efe Öge



benjaoming Benjamin Balder Bach

1) Generate the root encryption key



```
1 >> python manage.py generate_root_encryption_key --env development
2
3 Created root encryption key file: /development.env
4 Added development.env to /.gitignore
```

2) Encrypt a secret



```
1 >> python manage.py encrypt_secret --env development
2 Value to encrypt:
3 Plaintext: he*****ow
4 gAAAAABqvtcubb0F-
  BahQQUF5dPj56WdFrTH7wK7vEgyrZmz89GfLvdPr50ojQvecpEm105Kq0w-
  j9JvPEWAaNnu8lLX0AiJEQ=
```

3) Use the encrypted secret



```
1 # settings/development.py
2
3 from django_root_secret import get_secret
4
5 DATABASES = {
6     "default": {
7         ...
8         "PASSWORD": get_secret("gAAAAA..."),
9     }
10 }
```

Encrypted values travel with the code.

Only the root key must be provisioned separately.

We didn't eliminate secret management.

We reduced what has to be distributed.

An aerial photograph of a long bridge spanning a body of water. At the end of the bridge is a small, star-shaped island with a road and some vegetation. The water is a deep blue, and the sky is a lighter blue.

1) What is a secret?

2) SECRET_KEY in Django

3) Where should secrets live?

4) Who can read the secret?

5) Q&A

Who can read the secret?

- Django
- Celery workers
- CI/CD
- Deployment platform
- Developers
- Dependencies
- Logs / error reporting
- Coding agents / LLMs

Every reader becomes part of the trust boundary.

Secrets travel

.env



Django



traceback



copy / paste



LLM / Slack / issue

Most leaks don't require breaking cryptography.

What if the secret leaks?

Limit the capability

- A database credential shouldn't be a superuser
- An API credential should have only the scopes the app needs

Make replacement routine

- DB/API key: create new → deploy → revoke old
- Django SECRET_KEY: new key → old key in fallbacks → remove old key

Avoid long-lived secrets

- prefer an IAM role over AWS_SECRET_ACCESS_KEY
- Let the application obtain temporary credentials instead of storing a permanent key
- A leaked API credential should grant only what the app needs.

Secret checklist

Is it actually a secret?

Does it have only the capability it needs?

Was it generated securely?

Does it need to be long-lived?

Where does it live?

Who can read it?

Can I replace it?



Keeping secrets secret in Django

Efe Öge