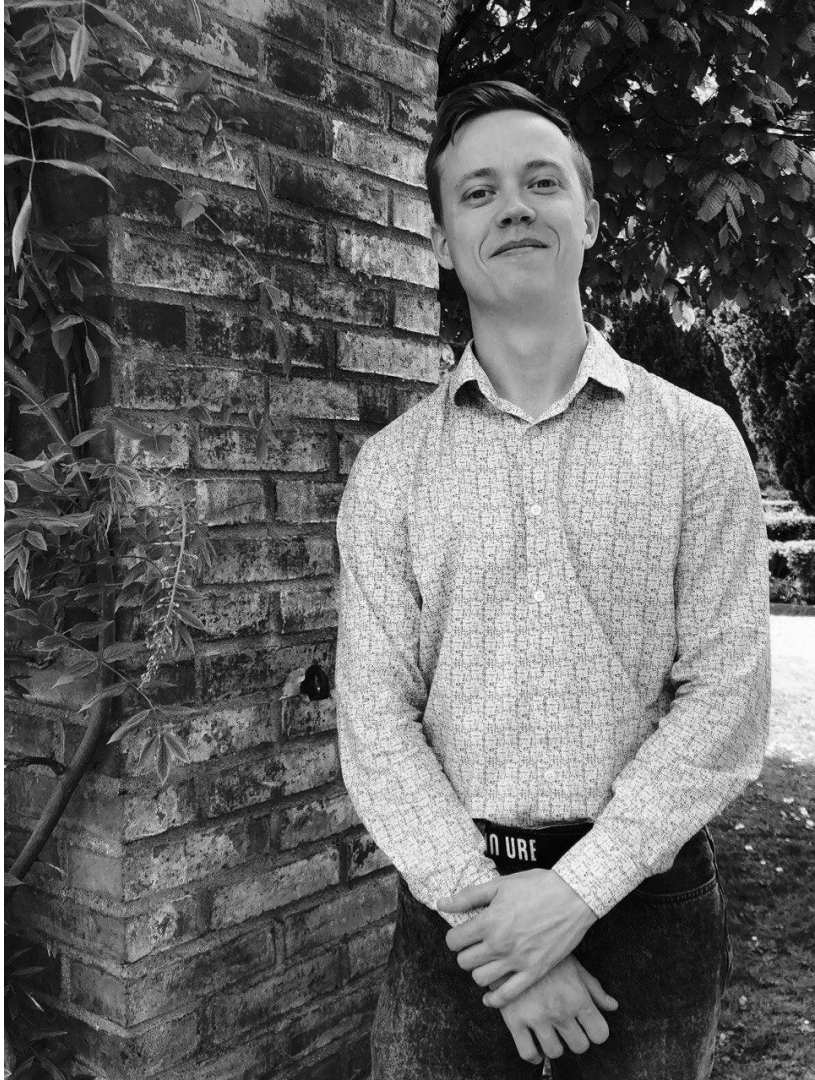

Protect sensitive data
with Symfony Secrets
by Viktor Pikaev

Viktor Pikaev

- Backend Developer and Tech Lead
- More than 16 years of experience
- A big fan of Symfony
- Establish Malmö Symfony Fans group

haru-atari.com

linkedin.com



Keep secrets in secret

No sensitive data like passwords, secret tokens and API keys should be stored as plain text in the code.

So no one can access it:

- people with access to repository
- developers
- github/gitlab

Variant 1
A separate config file

A separate config file is manually placed to the production server

```php
# config.php
return array_merge(
    [
        'dbName' => 'my-database',
    ],
    include __DIR__ . '/config-local.php',
);
```

```php
# config-local.php
return [
    'dbLogin' => 'my-login',
    'dbPassword' => 'my-password',
];
```

Variant 1
A separate config file

Pros

- It’s simple

Cons

- Requires manual updating
- Requires access to the production server

```php
# config.php
return array_merge(
    [
        'dbName' => 'my-database',
    ],
    include __DIR__ . '/config-local.php',
);
```

```php
# config-local.php
return [
    'dbLogin' => 'my-login',
    'dbPassword' => 'my-password',
];
```

Variant 2
Environment variables

The values are passed through environment variables during the deployment process (pipeline).

The values are stored in the CD system's secrets.

```php
# config.php
return [
    'dbName' => 'my-database',
    'dbLogin' => getenv('DB_LOGIN'),
    'dbPassword' => getenv('DB_PASSWORD'),
];
```

```
# .env
DB_LOGIN=db-login
DB_PASSWORD=db-password
```

Variant 2
Environment variables

Pros

- Allows us to automate configuration updating
- Allows for the secrets rotation

Cons

- Requires access to CD system's secrets to add new or change existing values

```php
# config.php
return [
    'dbName' => 'my-database',
    'dbLogin' => getenv('DB_LOGIN'),
    'dbPassword' => getenv('DB_PASSWORD'),
];
```

```
# .env
DB_LOGIN=db-login
DB_PASSWORD=db-password
```

Variant 3
Encrypted in-repo secrets

The values are stored in encrypted view in the repository.

They are decrypted on the production server with a private key.

The private key are stored in the CD system's secrets
and delivered during the deployment process.

Variant 3
Encrypted in-repo secrets

Pros

- Allows us to automate configuration updating
- Allows us to add new or change existing values without access to CD system's secrets
- Can be combined with passing values through environment variables

Cons

- Doesn't allow for secrets rotation

What to choose?

CD system's secrets for storing infrastructure secrets.

Encrypted in-repo secrets for storing application secrets.

Symfony Secrets

It’s a build-in implementation of the encrypted in-repo secrets.

It seamlessly integrates secrets to standard Symfony configuration system.

The application treats secrets like normal environment variables.

symfony.com

Symfony Secrets
How does it work?

1. Generate a pair of encryption keys.
2. Encrypt secrets with the public key.
3. Deliver the private key to production
4. Use the private key on production server to decrypt secrets.
5. Enjoy safety!

Step 0
Prepare the project

Move all sensitive values to the environment variables.

Symfony provides secrets like a normal environment variables.

```
# .env
MY_SECRET_TOKEN=adf2bj3hb234234
```

```
# parameters.yaml
parameters:
    token: '%env(MY_SECRET_TOKEN)%'
```

```
# bootstrap.php
use Symfony\Component\Dotenv\Dotenv;

require __DIR__ . '/vendor/autoload.php';
(new Dotenv(true))->load('.env');
```

Step 1
Generate the encryption keys

Generate encryption keys for each environment:

```
$ APP_RUNTIME_ENV={env}
$ php bin/console secrets:generate-keys
```

It generates a pair of keys:

```
config/secrets/{env}/{env}.decrypt.private.php
config/secrets/{env}/{env}.encrypt.public.php
```

Step 1
Generate the encryption keys

Add the production private key file to .gitignore. It should never be in the repo.

Use different key pairs for different environments.

Change nothing in the “config/secrets/*” directories manually. All files there are auto generated.

Step 2
Encrypt the secrets

Add a new or update an existing secret:

```
$ APP_RUNTIME_ENV={env}
$ php bin/console secrets:set {name}
```

Symfony will prompt the value, encrypt it and save it into the “config/secrets/{env}” directory.

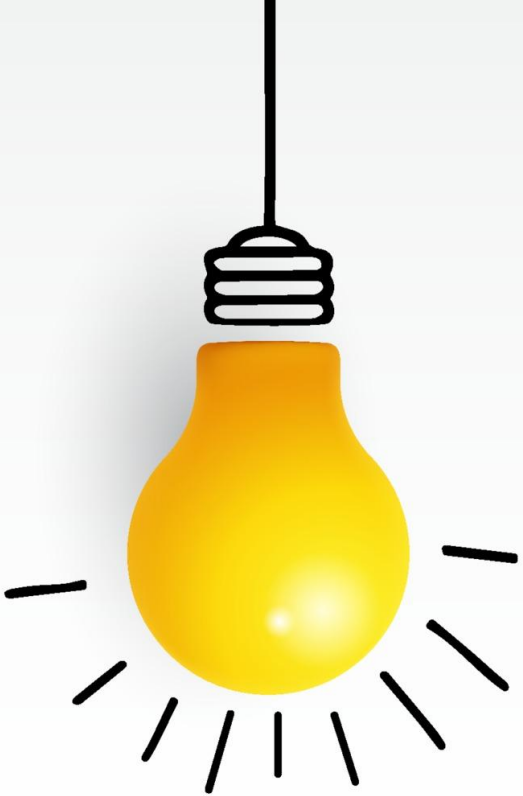
To remove an existing secret run:

```
$ APP_RUNTIME_ENV={env}
$ php bin/console secrets:remove {name}
```

Step 2
Encrypt the secrets

We do not need the private key for adding or updating secrets. So we can do it easily by ourselves.

No access requests, no waiting, no delais!



Step 2
Encrypt the secrets

Environment variables have higher priority than secrets. A secret will be overridden with the environment variable with the same name.

Step 3
Deliver the private key to production

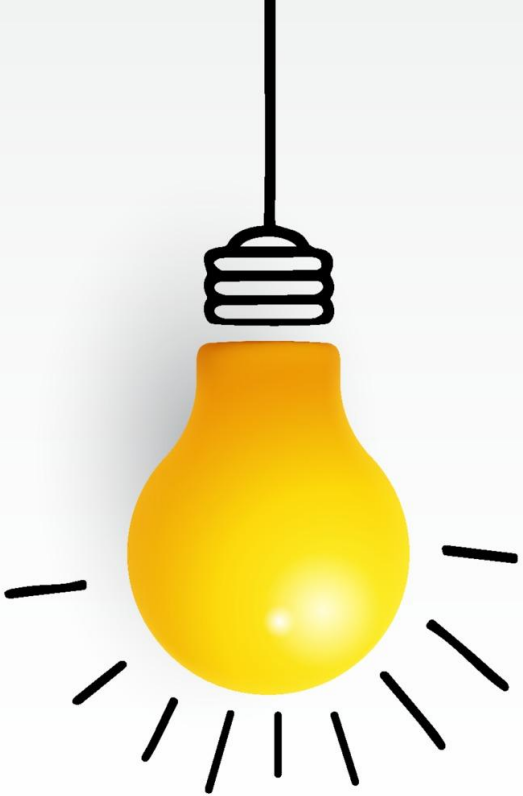
We have to keep the production private key safely out of the repository!

It should be delivered to production and be placed in the:
`config/secrets/prod/prod.decrypt.private.php`

Symfony will use it automatically. We don’t need to do anything else.

Step 3
Deliver the private key to production

You can pass the private key with the SYMFONY_DECRYPTION_SECRET environment variable instead of the file.



Step 4 (optional but strongly recommended)

Decrypt secrets on production

To decrypt all secrets and put them to the .env.prod.local file run:

```
$ php bin/console secrets:decrypt-to-local --force
```

- It allows non-Symfony php code to access secrets with the getenv() function.
- It gives us a small performance improvement. Small but nice.

Step 4
Decrypt secrets on production

Dotenv should be configured and work correctly to load values from the .env.prod.local file properly.

Step 5
Enjoy safety (but never relax)

You are awesome!

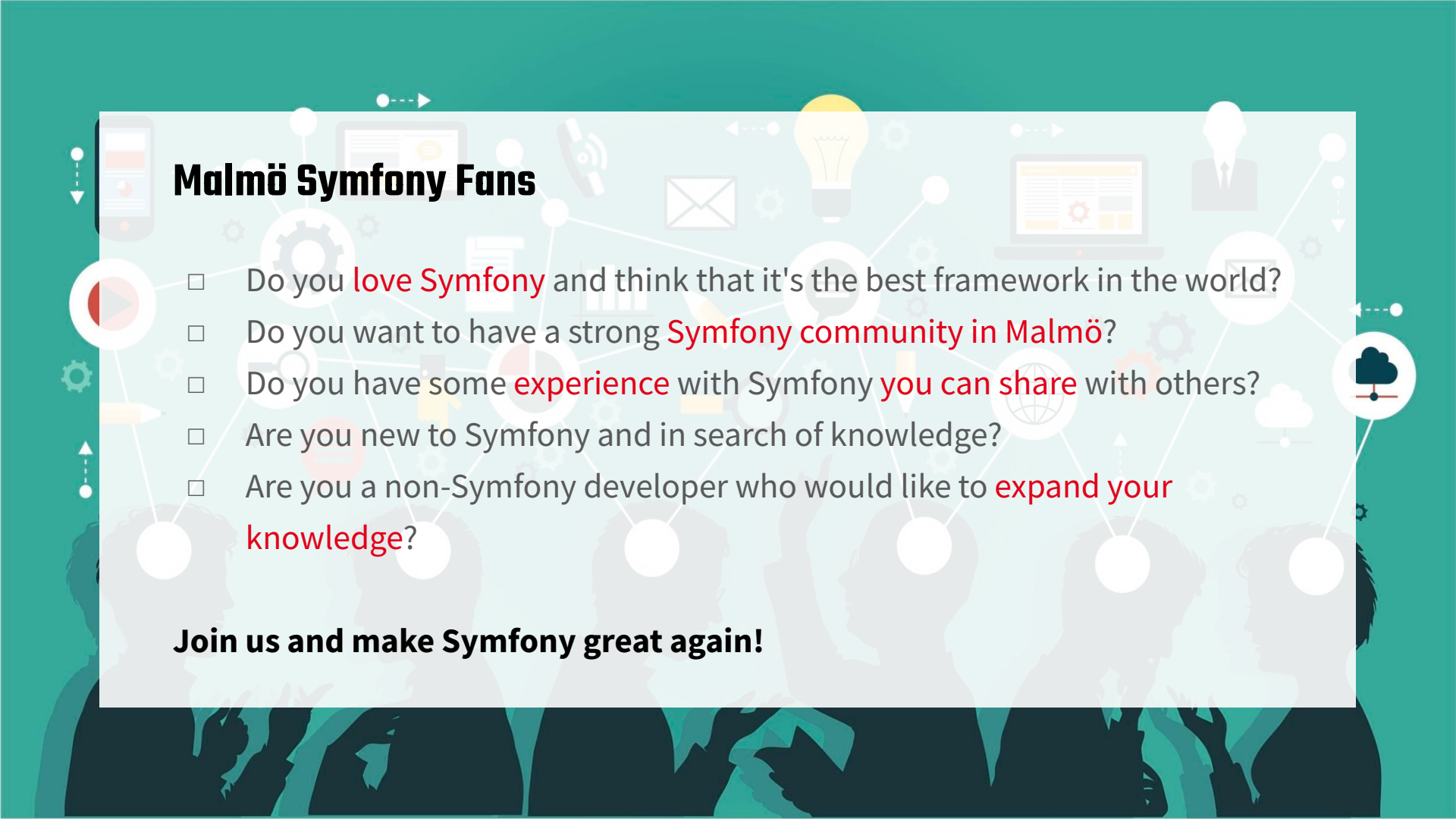
There are two pies on the shelf. Take the one in the middle. It’s yours.

What's next?

Trivy is the most popular open source security scanner, reliable, fast, and easy to use.

Add it to your pipeline to make sure all new secrets are stored properly.

trivy.dev


Malmö Symfony Fans
Do you love Symfony and think that it's the best framework in the world?
Do you want to have a strong Symfony community in Malmö?
Do you have some experience with Symfony you can share with others?
Are you new to Symfony and in search of knowledge?
Are you a non-Symfony developer who would like to expand your knowledge?
Join us and make Symfony great again!

slides

linkedin.com