Guidelines for the use of AGE

Fabio Massimo Ercoli *

November 15st 2024

1 Introduction

We're going to present a brief guide for using *AGE file encryption project*[1].

It is an easy to use library to encrypt/decrypt binary files using both symmetric and asymmetric keys algorithms. It is written in *Go* programming language, which is a popular language for programming on the cloud.

If you want to use the library, you can:

- Import the library into you source code project (a native support is offered for Go and Rust).
- Install it into operating systems, so that the API could be invoked from the operating system shell.

In this guide we will follow the later option.

1.1 Installation

As a first step we need to install Age in our local operating systems environment.

For instance in Fedora or Red Hat Linux you can simply run the command:

```
$ sudo dnf install age
```

For other operating system see the installation page [2].

To verify the correct installation type the following commands:

```
$ age --help
```

and

```
$ age-keygen --help
```

age and *age-keygen* are the API commands that can be invoked from the command line. The output provides the syntax for using those commands.

*More on me: https://github.com/fax4ever https://www.linkedin.com/in/fabioercoli/

[1] https://github.com/FiloSottile/age?tab=readme-ov-file

[2] https://github.com/FiloSottile/age?tab=readme-ov-file#installation

1.2 Features

Age does not limit in any way the types of file supported for the encryption / decryption. Thus we can assume that they are seen as binary files.

It only supports encryption / decryption, not signatures.

Compared with other tools (for instance OpenSSL) it looks easier to use, limiting for instance the possible algorithms to apply, but as we will see in the following sections, the algorithms are considered very secure at the moment.

I want also to mention that the library offers the capability to be extended by plugings. This is in nice since it provides more flexibility, but in general it is better to be sure about the security offered in particular by third party plugins before to use them.

2 Encryption and decryption

2.1 Passphrase encryption

Files can be encrypted and decrypted using the same (symmetric) passphrase. This is usually longer than a password, since it should contain more words, and this increases the size of the space of all the possible keys. I would say it is better than a common password, but, in my opinion, still not secure as a long binary key. Age by default auto generates the passphrase to use for the encryption, I would suggest to use this approach.

Use the option **-p** to use this strategy to encrypt a file:

```
$ age -p h1.pdf > h1.pdf.age
```

To decrypt option **-d** is always used

```
$ age -d h1.pdf.age > h1-decripted.pdf
```

2.2 Public key encryption

In the library language:

- a **recipient** is a public key (or more in general a public value).
- an **identity** is a private key (or more in general a private value).

The former is used to encrypt, the latter is used to decrypt.
For public key encryption you can specify one or more recipient using the options:

- **-r** Providing directly the value of the recipient (more -r can be used in the same encryption)
- **-R** Providing the path to the file containing the recipient(s).

For private key decryption you can specify one or more identities using the options:

- **-i** Providing directly the value of the identity (more -i can be used in the same encryption)
- **-I** Providing the path to the file containing the identity(s).

You can use two kind of key pairs:

- **Native X25519 keys**, using elliptic curve Diffie-Hellman key exchange. The elliptic curve techniques in general allow to keep high the security with keys having a relatively small size.
- **SSH keys** generated with other tools.

You can use directly the library to generate Native X25519 keys, for instance running the command:

```
$ age-keygen -o key.txt
```

Opening the *key.txt* file, you should find a content similar to the following:

```
# created: 2024-11-14T11:58:08+01:00
# public key: age1x864sfs5pfka87tgpaqn0j0lc89ep3cw35sr69et9dz2ncwf6uaskmzv8w
AGE-SECRET-KEY-14KG6Z6VR72EA2YG6J56Q9LS5WV6Z825SJ4VZUV5JWTL4VM9HJK4SZAFYG2
```

Here is an example of encryption:

```
$ age -r age1x864sfs5pfk... h2.pdf > h2.pdf.age
```

Here is an example of decryption:

```
$ age -d -o h1-decripted.pdf -i key.txt h2.pdf.age
```

3 Algorithms

We already mentioned that native keys generated by AGE are generated according to the standard *X25519*.

For symmetric encription an *AEAD* (authenticated encryption with associated data) solution is used, in particular AGE uses *ChaCha20* stream cipher that is used tougher with the *Poly1305* authenticator. This solution is usually faster than the more classical *AES-GCM AEAD*.

Keys are generated from passphrases using *scrypt*, that is a password-based key derivation function algorithm. In all the other cases *HKDF-SHA-256*, a key derivation algorithm, is used to derive the keys.