Keyoxide: *Bob, is that you?*

Backstory

Keyoxide is an open-source platform for verifying digital identities. No, it is not CIE, it is not SPID, eIDAS, or whatever you like: it is none of those. It does not serve to prove that a person is who they are reported to be on an ID document like a driver's license, but it certifies that whoever claims ownership of something (a GitHub repository, an identity on Lemmy, etc.) is indeed the holder of the private key with which their digital signature was created.

Put in more classic terms:

Keyoxide is an open-source platform for decentralized digital identity validation. Unlike federated identity systems (such as SPID or CIE) or centralized authentication protocols (OAuth), Keyoxide does not act as a civil registry certification authority, but rather as an aggregator of cryptographic proofs.

The core of the software is based on verifying ownership rather than civil identity. Using cryptographic standards such as OpenPGP or the ASPE protocol, Keyoxide allows a user to uniquely link different digital properties (code repositories, social accounts, web domains) to a single cryptographic root.

"... *But then in the end... what a mess! Everyone looking... the toad looking at the... the turkey looking at the bitch... I say: "Hold still, or what's gonna be born, the... the tur-bitch?! Be good..!". Who was mixing here... then, you just had to get distracted, they were screwing everything! They were screwing everyone! Someone screwing here, someone screwing there..! The poor mole was blind, she couldn't see, they all screwed her! I mean really everyone! You should have seen what they did to her! Every now and then she noticed something was wrong, it was too much! So she would ask: "Excuse me, but are you the male mole?" – "Yeah, yeah... I'm the male mole!".. and... just absolute pigs!* ...."

This is taken from the show "Tutto Benigni 95/96", and it is a clear example that Mrs. Mole could probably have benefited from Keyoxide to

be sure that whoever was behind it was indeed the male mole.

Use Case

You create an identity on a social network like Mastodon, and then on Lemmy, then on GitHub, then you realize that maybe your social life would be better off oriented towards the outside world, but there you go. In any case, now in total depression, it also occurs to you that people probably don't know that these three identities are linked together and can always be traced back to you. How do you reassure them, as if their only concern were knowing whether the user *Ciottolino94* on GitHub is the same *Ciottolino94* whose post they saw on Mastodon? The name is quite embarrassing, so I think very few people have the guts to choose this exact username, but if you want to be absolutely sure, what do you do?

- Create a cryptographic key with GPG (you can also use the ASPE protocol, but now we'll only cover the PGP version which is INCOMPATIBLE with ASPE)
- Associate an email with it
- On the various services you use, insert an Ariadne Proof, which helps Keyoxide understand that whoever created that page is the same person who holds the signature
- Anyone who wants to verify checks on Keyoxide.

Let's guide: Step 1, GPG

Install GPG, though I hope you already have it. Then create a signing key, preferably elliptic.

In the examples below, I am creating a key for *admin@example.org*.

```
gpg --expert --full-generate-key
gpg (GnuPG) 2.4.9; Copyright (C) 2025 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
```

```
   (7) DSA (set your own capabilities)
   (8) RSA (set your own capabilities)
   (9) ECC (sign and encrypt) *default*
  (10) DSA (sign only)
  (11) ECC (set your own capabilities)
  (13) Existing key
  (14) Existing key from card
Your selection? 9
Please select which elliptic curve you want:
   (1) Curve 25519 *default*
   (2) Curve 448
   (3) NIST P-256
   (4) NIST P-384
   (5) NIST P-521
   (6) Brainpool P-256
   (7) Brainpool P-384
   (8) Brainpool P-512
   (9) secp256k1
Your selection? 1
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify the key.

Real name: Permission Denied
Email address: admin@example.org
Comment:
You selected this User ID:
    "Permission Denied <admin@example.org>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, use the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, use the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: revocation certificate stored as '/home/errno0x0d/.gnupg/openpgp-
```

```
revocs.d/3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9.rev'
public and secret keys created and signed.

pub   ed25519 2026-04-06 [SC]
      3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9
uid                      Permission Denied <admin@example.org>
sub   cv25519 2026-04-06 [E]

```

Step 2: Modifying the key

Take a good look at the 40 characters of the fingerprint `3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9`: you'll need it later:

```
pub   ed25519 2026-04-06 [SC]
      3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9
uid                      Permission Denied <admin@example.org>
sub   cv25519 2026-04-06 [E]
```

Now: what do you want to certify? The available claims are here (https://docs.keyoxide.org/service-providers/).

Let's take Lemmy (https://docs.keyoxide.org/service-providers/lemmy/).

We edit the key, select *uid 1* (usually there's only that one), add the *notation* for the Ariadne proof, and save:

```
gpg --edit-key 3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9
gpg (GnuPG) 2.4.9; Copyright (C) 2025 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   9  signed:   2  trust: 0-, 0q, 0n, 0m, 0f, 9u
gpg: depth: 1  valid:   1  signed:   0  trust: 1-, 0q, 0n, 0m, 0f, 0u
gpg: next trustdb check due at 2029-01-21
sec  ed25519/E035163A2EC53669
     created: 2026-04-06  expires: never       usage: SC
     trust: ultimate      validity: ultimate
ssb  cv25519/7249B111D145D261
```

```
     created: 2026-04-06  expires: never       usage: E
[ultimate] (1). Permission Denied <admin@example.org>

gpg> uid 1

sec  ed25519/E035163A2EC53669
     created: 2026-04-06  expires: never       usage: SC
     trust: ultimate      validity: ultimate
ssb  cv25519/7249B111D145D261
     created: 2026-04-06  expires: never       usage: E
[ultimate] (1)* Permission Denied <admin@example.org>

gpg> notation
Enter the notation: proof@ariadne.id=https://lemmy.example.org/u/errno0x0d
No notations on user ID "Permission Denied <admin@example.org>"
Adding notation: proof@ariadne.id=https://lemmy.example.org/u/errno0x0d

sec  ed25519/E035163A2EC53669
     created: 2026-04-06  expires: never       usage: SC
     trust: ultimate      validity: ultimate
ssb  cv25519/7249B111D145D261
     created: 2026-04-06  expires: never       usage: E
[ultimate] (1)* Permission Denied <admin@example.org>

gpg> save
```

Step 3: Key exporting and publishing

We need to upload the public key to keys.openpgp.org to make it visible to Keyoxide:

```
gpg --armor --export 3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9 > key.asc
```

After that, we go to `https://keys.openpgp.org/upload/` and upload the key. We will also be told that to make it visible we need to verify the email that will be sent to us (to prove that the mailbox exists and is ours). Let's do it.

Step 4: Adding the Ariadne proof

In Lemmy's case, it is sufficient to go to your profile and insert the following string in the bio (note the 40 characters of the

fingerprint):

```
openpgp4fpr:3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9
```

You can also see it by visiting my profile here (https://lemmy.example.org/u/errno0x0d).

Step 5: Verification

By going to `https://keyoxide.org/3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9` (as you can see, the key fingerprint has been inserted), you can see which digital identities are associated with the user.

Step 6: Next times

If in the future we want to send key updates directly from the terminal without going through the browser, we can configure our `~/.gnupg/gpg.conf` by adding `keyserver hkps://keys.openpgp.org` and use: `gpg --send-keys 3AC3F8D6A11B6CED3330DB8BAE1C2D88ADD99AF9`.

For each proof we add, steps 2-6 must be repeated.

[May the big mole be with you]

[EOF]