Using elliptic curve cryptography for the purposes of online identity

Yarmo Mackenbach <yarmo@keyoxide.org>

February 3rd, 2024 @ FOSDEM

https://codeberg.org/yarmo/fosdem-2024

How to verify a claim?

With a proof!

An example of claim and proof

- Alice lost their luggage
- Bob found the luggage
- Alice claims the luggage is theirs
- Bob asks for proof
- Alice unlocks the luggage

Claim verified!

Can we claim and prove over the internet?

Yes!

Using cryptographic signatures.

What is a cryptographic signature?

Signature	Pen-and-paper	Cryptographic
Purpose?	Acknowledge a statement	Acknowledge a statement
Forgery?	Simple	Practically impossible
Signer?	Person	Secret key
Verification?	Visual inspection	Public key

In short:

Secret key $\rightarrow$ sign documents
Public key $\rightarrow$ verify signatures

Secret key + public key = keypair

Each keypair has a unique **fingerprint**

An example of *online* claim and proof

- I write a text document:

```
My Fediverse account is https://fosstodon.org/@yarmo
```

- I sign it with my secret key (fingerprint: `FOSSXQF6Z4POZG`):

```
01100110011011101110011[…] # don't worry about this
```

- I give the document and signature to my friend
- My friend verifies the signature ✓
- My friend visits my Fediverse account and reads in the bio:

```
My key fingerprint is FOSSXQF6Z4POZG
```

Claim verified!

Creating an identity claim in 100 lines of Rust

Rust dependencies

```
data-encoding
josekit
reqwest
serde_json
sha2
```

You could do without these, but...

Generate a cryptographic key

```
1  // EC = Elliptic Curve
2  // P256 = a specific curve of EC
3
4  let curve = EcCurve::P256;
5  let key = EcKeyPair::generate(curve)?
6    .to_jwk_key_pair();
```

Get the key's fingerprint

```
// Collect information about the key
let key_json = json!({
  "crv": key.parameter("crv")?.as_str(),
  "kty": key.parameter("kty")?.as_str(),
  "x": key.parameter("x")?.as_str(),
  "y": key.parameter("y")?.as_str(),
});

// Hash the information
let mut hasher = Sha512::new();
hasher.update(key_json.to_string().as_bytes());
let hash: [u8; 64] = hasher.finalize().into();

// Encode the hash
let fingerprint = BASE32_NOPAD.encode(&hash[0..16]);
```

Create an identity profile

Identity profile: has a name and one or multiple claims

```
let name = "Yarmo";
let claims = vec![
  "https://fosstodon.org/@yarmo"
];
```

Create a JSON Web Token for the profile

JWT: combine a text document and a cryptographic signature.

Three parts:

- a header
- a payload
- the signature

Some notes before we continue

Note: `http://ariadne.id/...` is the namespace we use.

Note: You may see *jws* as well sometimes. Don't worry about it...

Create a JSON Web Token for the profile

```
// Create a header

let mut header = JwsHeader::new();
header.set_key_id(fingerprint);
header.set_jwk(key.to_public_key()?);
// [...]
```

Create a JSON Web Token for the profile

```
// Create a payload

let mut payload = JwtPayload::new();

payload.set_claim(
  "http://ariadne.id/type",
  Some(json!("profile")))?;

payload.set_claim(
  "http://ariadne.id/name",
  Some(json!(name)))?;

payload.set_claim(
  "http://ariadne.id/claims",
  Some(json!(claims)))?;
// [...]
```

Create a JSON Web Token for the claim

```
1  // Sign the header and the payload
2
3  let signer = Es256.signer_from_jwk(&key)?;
4  let profile_jwt = jwt::encode_with_signer(
5    &payload, &header, &signer)?;
```

Our profile is ready!


```
eyJ0eXAiOiJKV1QiLCJraWQiOiJXUONMSTNRQUFKUElYMk1IMkJW
NVFBMlgOTSIsImp3ayI6eyJrdHkiOiJFQyIsImNydiI6IlAtMjU2
IiwieCI6ImlmSGVRMXctWmpVNUdlZVRLSWxhc1B6ejRYLWJuYV9n
eW9QamVUX05OeDAiLCJ5IjoieU5ueTR6MmZVMWdUZTRQTWVDV2t4
aEFzSi1zWVJNZFlLU3FoNm1kbDhTQSJ9LCJhbGciOiJFUzI1NiJ9
.eyJodHRwOi8vYXJpYWRuZS5pZC92ZXJzaW9uIjowLCJodHRwOi8
vYXJpYWRuZS5pZC90eXBlIjoicHJvZmlsZSIsImh0dHA6Ly9hcml
hZG5lLmlkL25hbWUiOiJBbGljZSIsImh0dHA6Ly9hcmlhZG5lLml
kL2Rlc2NyaXB0aW9uIjoiVGhpcyBwcm9maWxlIHdhcyBjcmVhdGV
kIGxpdmUgb24gc3RhZ2UgYXQgRk9TREVNIDIwMjQgOikiLCJodHR
wOi8vYXJpYWRuZS5pZC9jbGFpbXMiOlsiaHR0cHM6Ly9mb3NkZW0
ub3JnIl0sImlhdCI6MTcwNjg3OTc2Nn0.E4E8uxmpFawh46ot55R
Wlx94ca8zo7mKNiwm4QE1-YmDmCLACvqA8tsCcZRqDtS6F52MHOM
4A_NHce7NBTl8Bw
```

A JWT inside a JWT...

Create a JWT to upload the profile

```
1 // Re-use the header from before
```

Create a JWT to upload the profile

```
// Create a payload

let mut payload2 = JwtPayload::new();

payload2.set_claim(
  "http://ariadne.id/type",
  Some(json!("request")))?;

payload2.set_claim(
  "http://ariadne.id/action",
  Some(json!("create")))?;

payload2.set_claim(
  "http://ariadne.id/profile_jws",
  Some(json!(profile_jwt)))?;
// [...]
```

Create a JWT to upload the profile

```
1  // Sign the header and the payload
2
3  let request_jwt = jwt::encode_with_signer(
4    &payload2, &header, &signer)?;
```

Upload the profile

```
// Make HTTP POST request to server

let host = "dev.keyoxide.org";
let endpoint = "/.well-known/aspe/post";
let url = format!("https://{}{}", host, endpoint);

let client = reqwest::blocking::Client::new();
let req = client
  .post(url)
  .header(reqwest::header::CONTENT_TYPE,
    "application/asp+jwt")
  .body(request_jwt)
  .send()?;
```

Do try this at home!

Thanks!

About Keyoxide

https://keyoxide.org
Join the community on Matrix & IRC :)

Fully open source

https://codeberg.org/keyoxide
Project is growing, looking for new contributors!

Huge thanks to

NLnet Foundation & NGI0
OpenCollective supporters
All the contributors