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?

<table>
<thead>
<tr>
<th>Signature</th>
<th>Pen-and-paper</th>
<th>Cryptographic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose?</td>
<td>Acknowledge a statement</td>
<td>Acknowledge a statement</td>
</tr>
<tr>
<td>Forgery?</td>
<td>Simple</td>
<td>Practically impossible</td>
</tr>
<tr>
<td>Signer?</td>
<td>Person</td>
<td>Secret key</td>
</tr>
<tr>
<td>Verification?</td>
<td>Visual inspection</td>
<td>Public key</td>
</tr>
</tbody>
</table>

**In short:**

- Secret key → sign documents
- Public key → 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):
  ```
  011001100110111101110011[…][... # 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

1. data-encoding
2. josekit
3. reqwest
4. serde_json
5. sha2

You could do without these, but...
Generate a cryptographic key

```rust
// EC = Elliptic Curve
// P256 = a specific curve of EC

let curve = EcCurve::P256;
let key = EcKeyPair::generate(curve)? .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

```rust
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

```rust
// 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

```rust
// 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

// Sign the header and the payload

let signer = Es256.signer_from_jwk(key));
let profile_jwt = jwt::encode_with_signer(
    &payload, &header, &signer);
Our profile is ready!
A JWT inside a JWT...
Create a JWT to upload the profile

// Re-use the header from before
Create a JWT to upload the profile

```rust
// 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

```rust
// Sign the header and the payload
let request_jwt = jwt::encode_with_signer(
    &payload2, &header, &signer)?;
```
Upload the profile

```rust
// 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