Web eID authentication token format specification

Mart Sõmermaa, Estonian Information System Authority

February 4, 2022

1 Introduction

As of September 2021, the Estonian Information System Authority is preparing to introduce Web eID – a new architecture solution for web authentication and signing[1]. In this new architecture, a user of the Estonian ID card is authenticated to a website on the application level by signing the website's challenge with the help of the Web eID browser extension.

Until now, Web eID is using the OpenID X509 ID Token format[2] in authentication tokens. However, the current authentication token format has weaknesses that make it possible to use it incorrectly.

In this paper, we provide an overview of the weaknesses, and the specification of the new Web eID authentication token format that mitigates the weaknesses.

The paper is based on Arnis Paršovs paper *On the format of the authentication proof used by RIA's Web eID solution*[3].

2 The Web eID authentication token format

The current Web eID authentication token format is based on the OpenID Connect ID Token[4] specification. All fields required in the OpenID Connect specification are present, including `iat`, `exp`, `iss` and `sub` that are ignored by the Web eID authentication protocol.

Example token:

`Header:`

[1] https://github.com/web-eid/web-eid-system-architecture-doc

[2] https://github.com/web-eid/web-eid-system-architecture-doc#openid-x509-id-token-specification

[3] https://cybersec.ee/storage/webeid_auth_proof.pdf

[4] http://openid.net/specs/openid-connect-core-1_0.html#IDToken

```
{
  "typ": "JWT",
  "alg": "RS256",
  "x5c": ["MIIFozCCA4ugAwIBAgIQHFpdK-zCQsFW4..."]
}
```

`Payload:`

```
{
  "aud": ["https://ria.ee"],
  "exp": "1479621923",
  "iat": "1479621900",
  "iss": "web-eid app v1.0.0",
  "sub": "John Doe",
  "nonce": "NONCEVALUE"
}
```

`Signature: ...`

The header part contains a JSON structure that contains the `alg` field which identifies the cryptographic algorithm used to create the signature and the `x5c` field which contains the user's authentication certificate. The payload part contains a JSON structure depicted above and is signed together with the header using the user's authentication key. The signature part carries the value of the user's signature.

2.1 Weaknesses of the current OpenID X509 ID Token format

The only values that have to be included under the user's signature to achieve the security properties of the protocol are the website's challenge (the `nonce` field above) and the website's origin (the `aud` field above). The inclusion of the fields `exp`, `iat`, `iss` and `sub` under the signature serve no practical purpose. On the contrary, the presence of these fields in the authentication token introduces a risk of vulnerabilities in case the authentication implementation of a website decides to rely on any of them for making security critical decisions. A correct implementation should ignore these fields and verify the freshness of the authentication token using a locally-stored trusted timestamp that indicates the time when the challenge was issued.

While the fields `nonce` and `aud` must be included under the signature, including them in the authentication token introduces a risk of man-in-the-middle relay impersonation attacks, as a faulty implementation can verify the signature without ensuring that the fields included under the signature correspond to the trusted values stored locally by the website.

Furthermore, the inclusion of the `nonce` field in the authentication token introduces a risk of forged login attacks, as a faulty implementation may use the nonce

value from the received authentication token to lookup the corresponding data in its local storage, without verifying that the authentication token is received from the same browser to which the corresponding challenge was issued. Such a flaw would enable a cross-site request forgery attack where an attacker can forge a request to force a victim's browser to log into a vulnerable website using the attacker's credential (authentication token).

Even though the Web eID project provides ready-made libraries for validating the authentication token securely, it is possible that not every developer implementing the solution is able to use them. In this case it is also possible that they will not closely examine the documentation and will not be able to precisely follow the instructions to ignore certain fields in the token. Therefore, it is desirable to design a security protocol in a manner that makes implementation mistakes less likely to occur.

2.2 Reasoning behind the current format

As mentioned above, the redundant fields `exp`, `iat`, `iss` and `sub` have been included in the JWT authentication token to support compatibility with the OpenID Connect ID Token specification. The security analysis of the Web eID solution further adds that the use of the OpenID Connect format offers a cheaper migration path, as the format is already known for e-service developers[5].

We note that this reasoning is flawed because it is not possible to achieve compatibility and integration between two conceptually different solutions just by making the data exchange format used by the solutions look the same. The purpose of OpenID Connect (and JWT in general) is to exchange identity claims that are signed by a trusted party (usually an authentication server), while the purpose of the Web eID authentication token is to prove that the user is able to create signatures with the private key that corresponds to the presented certificate.

We argue that any similarities of the Web eID authentication token to the JWT format are actually undesirable, as they would imply that the claims presented in the Web eID authentication token can be trusted and processed, while actually they must be ignored. For the same reason the use of the current format of the authentication token cannot provide a cheaper migration path, because the same codebase or workflow that is applied to any other JWT must not be applied to the Web eID authentication token, to not introduce security vulnerabilities or other unintended behavior.

[5] https://web-eid.github.io/web-eid-cybernetica-analysis/webextensions-main.pdf, section 3.1.

2.3 Proposed new format for the Web eID authentication token

Since to our knowledge there does not exist a standardized format for an authentication proof that implements nothing less and nothing more than is necessary for the Web eID authentication protocol, we propose to use a simple and foolproof[6] special purpose format for the Web eID authentication token. We intentionally avoid using the JWT format, but still use its proven basic building blocks: the JSON format and base64-encoding.

The Web eID authentication token is a JSON data structure that looks like the following example:

```
{
  "unverifiedCertificate": "MIIFozCCA4ugAwIBAgIQHFpdK-zCQsFW4...",
  "algorithm": "RS256",
  "signature": "HBjNXIaUskXbfhzYQHvwjKDUWfNu4yxXZha...",
  "format": "web-eid:1.0",
  "appVersion": "https://web-eid.eu/web-eid-app/releases/v2.0.0"
}
```

It contains the following fields:

- `unverifiedCertificate`: the base64-encoded DER-encoded authentication certificate of the eID user; the public key contained in this certificate should be used to verify the signature; the certificate cannot be trusted as it is received from client side and the client can submit a malicious certificate; to establish trust, it must be verified that the certificate is signed by a trusted certificate authority,
- `algorithm`: the signature algorithm used to produce the signature; the allowed values are the algorithms specified in JWA RFC[7] sections 3.3, 3.4 and 3.5:

  ```
  "ES256", "ES384", "ES512", // ECDSA
  "PS256", "PS384", "PS512", // RSASSA-PSS
  "RS256", "RS384", "RS512"  // RSASSA-PKCS1-v1_5
  ```

- `signature`: the base64-encoded signature of the token (see the description below),
- `format`: the type identifier and version of the token format separated by a colon character ':', `web-eid:1.0` as of now; the version number consists of the major and minor number separated by a dot, major version changes are incompatible with previous versions, minor version changes are backwards-compatible within the given major version,

[6] So simple, plain, or reliable as to leave no opportunity for error, misuse, or failure.
[7] https://www.ietf.org/rfc/rfc7518.html

- `appVersion`: the URL identifying the name and version of the application that issued the token; informative purpose, can be used to identify the affected application in case of faulty tokens.

The value that is signed by the user's authentication private key and included in the `signature` field is `hash(origin)+hash(challenge)`. The hash function is used before concatenation to ensure field separation as the hash of a value is guaranteed to have a fixed length. Otherwise the origin `example.com` with challenge `.eu1234` and another origin `example.com.eu` with challenge `1234` would result in the same value after concatenation. The hash function `hash` is the same hash function that is used in the signature algorithm, for example SHA256 in case of RS256.

To verify the signature, the website has to reconstruct the signed data. Since the challenge value and the origin field are not included in the token in the proposed solution, the website is forced to reconstruct the signed data using the origin and challenge values from its trusted local storage. This provides an important security advantage as it is guaranteed that if the signature verification succeeds, then the origin and challenge have been implicitly and correctly verified without the need to implement any additional security checks. Furthermore, it also guarantees that the authentication proof was received from the same browser to which the corresponding challenge was issued, as the website is forced to lookup the challenge and, possibly, the origin, in case it can vary, from its local storage using an identifier specific to the browser session.