On the format of the authentication proof used by RIA's Web eID solution

Arnis Parsovs
University of Tartu

October 14, 2021

1 Introduction

As of September 2021, the Estonian Information System Authority (Riigi Infosüsteemi Amet – RIA) 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. The designers of Web eID have chosen to use the JSON Web Token (JWT) format to carry the authentication proof signed by the user. In our opinion, the use of JWT in this context is a bad design choice as it introduces a list of security risks. In this paper we explain the problems with the current format and suggest a more foolproof design for the authenticaiton proof.

2 Security requirements for authentication

To securely authenticate a user, in a Web eID authentication process a website has to gain assurance that a user who claims to be the holder of the presented ID card authentication certificate has access to the corresponding private key. This is achieved by the website generating a random challenge and verifying that the user can provide a signature over it using their private key. Since a random challenge is unique and unpredictable, the website can assure that the signature is fresh, i.e., that the signature could have been made only after the challenge was issued.

However, since other websites are also eligible to request a signature over a challenge, the website cannot ensure that the signature over its challenge is provided by the user who initiated authentication to their website. Such a signature could also have been made while the user authenticated in some other website which simply passed the challenge from the original website to the user and the signed challenge back to the original website. To prevent such man-in-the-middle relay impersonation attacks [2], the user's signature has to be bound to the identity of the website from which the authentication challenge was received. In this process it is crucial that: (i) the identity of the website is included under the signature by the browser extension based on the actual website from which the authentication challenge was received, and (ii) the website that receives the signature verifies that the website's identifier under the signature matches the identity of the website. In the context of Web eID, the identity of the website is called an origin and in practice it corresponds to the domain name of the website.

To summarize, the only fields that have to be included under the signature in order to achieve the required security properties of the Web eID authentication protocol are the challenge (set by the website) and the origin (set by the browser extension, known by the website).

The Web eID specification [1] also discusses binding the user's signature to the Token Binding ID provided by the TLS Token Binding extension [3]. Such a binding would prevent man-in-the-middle impersonation attacks even in the case of a powerful attacker that is able to obtain a valid TLS certificate in the name of the website (e.g., due to a failure of Web PKI). However, since there are no indications that this feature will be widely supported by browsers and servers any time soon, we have excluded it from the consideration. We note that the question of whether to include the Token Binding ID under the signature does not have a significant effect to the matter discussed in this paper.

3 Current format of the authentication proof

According to the current Web eID specification, after a website calls the `webeid.authenticate()` method and a user successfully signs the authentication proof, the `webeid.authenticate()` method returns the JWT data structure, depicted in Figure 1, to the website.

```
Header:
{
 "typ": "JWT",
 "alg": "RS256",
 "x5c": ["MIIFozCCA4ugAwIBAgIQHFpdK..."]
}
Payload:
{
 "exp": "1479621923",
 "iat": "1479621900",
 "aud": "https://foobar.example.com/",
 "iss": "https://self-issued.me",
 "sub": "EE:38207162722",
 "nonce": "NONCEVALUE",
 "cnf": {
     "tbh": "l1X0aVlpikNqDhaH92VwGgrFdAY0tSackYis1r_-fPo"
  }
}
Signature:
...
```

Figure 1: Authentication proof returned by `webeid.authenticate()`

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.

As discussed in Section 2, 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), the website's origin (the `aud` field above) and the Token Binding ID (the `tbh` field above) if it is supported. The inclusion of the fields `exp`, `iat`, `iss` and `sub` under the signature **serve no practical purpose**. On the contrary, we argue that **the presence of these**

fields in the authentication proof 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. The most dangerous is the `sub` field that is filled with the supposed personal identification code of the user. If a developer who implements verification of the authentication proof fails to recognize that the value in this field must not be trusted (even though it is included under the signature), then the website's implementation will become vulnerable to a trivial authentication bypass flaw as the user who signs the proof can include an arbitrary personal identification code in this field. The same applies to the fields `iat` and `exp` that are supposed to indicate the time when the authentication proof was signed and when it should expire. A correct implementation should ignore these fields and instead verify the freshness of the authentication proof using a locally-stored trusted timestamp that indicates the time when the challenge was issued or the time when the user initiated the authentication process.

While the fields `nonce`, `aud` and `tbh` must be included under the signature, **including them in the authentication proof 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 proof **introduces a risk of forged login attacks**, as a faulty implementation may use the `nonce` value from the received authentication proof to lookup the cor responding data in its local storage, without verifying that the authentication proof 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 credentials (authentication proof).

It is unreallistic to assume that each and every developer implementing the solution will closely examine the documentation and will be able to precisely follow the counter-intuitive instructions given therein. Therefore, it is desirable to design a security protocol in a manner that makes implementation mistakes less likely to occur.

3.1 Reasoning behind the current format

According to the Web eID specification [1], the unnecessary fields `exp`, `iat`, `iss` and `sub` have been included in the JWT authentication proof to support integration with the OpenID Connect 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 (Section 3.1 in [4]).

We note that this reasoning is flawed because **it is not possible to achieve compatability 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 proof 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 proof to the JWT format are actually undesirable**, as they would imply that the claims presented in the Web eID authentication proof can be trusted and processed, while actually they must be ignored. For the same reason **the use of the current format of the authentication proof 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 proof, to not introduce security vulnerabilities or other unintended behavior.

If any kind of standards-compliance or integration with the existing libraries is desired, then the closest format to be used for the authentication proof could be JSON Web Signature (JWS) [5]. However, we would advice against it, because it is a general purpose data format for exchanging signed data and thus would still require modifications to achieve the desirable security properties for the Web eID authentication proof (see the next section).

4 Proposed format for the authentication proof

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[1] special purpose format for the Web eID authentication proof.

We propose to modify the `webeid.authenticate()` method such that it returns only three values: `certificate`, `signature` and `alg`. The `certificate` value contains the authentication certificate of the user, the `signature` value contains the signature that can be verified using the certificate, and the `alg` value contains one of the whitelisted signature algorithms that can be used to verify the signature.

The value that is signed by the user's authentication private key is `SHA256(origin)+SHA256(challenge)`. The hash function is used here to ensure field separation as the hash of a value is guaranteed to have a fixed length. Of course, the values `origin` and `challenge` could instead be encoded using a JSON structure, but introducing a dependency on a JSON encoder for combining two values might not be well justified.

To verify the signature, the website has to reconstruct the signed data. Since in the proposed solution the challenge value and the origin field is not returned by the `webeid.authenticate()` method, 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, the origin and challenge has 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 from its local storage using the browser's session identifier.

As an additional security improvement to the current Web eID API, we recommend removing the option to retrieve the authentication certificate using the `webeid.getCertificate()` method[2], as **this option introduces the risk**

[1]So simple, plain, or reliable as to leave no opportunity for error, misuse, or failure.

[2]Or alternatively, remove the certificate from the `webeid.authenticate()` response.

of a resource confusion vulnerability that can lead to an authentication bypass flaw. More specifically, there is a risk of a website authenticating a user using the certificate obtained through the `webeid.getCertificate()` call, while the authentication proof signed using some other key is successfully verified using a different certificate returned by the `webeid.authenticate()` call.

5 Further hardening of the protocol

We advise RIA to further harden the Web eID authentication protocol to ensure that a successful signature verification of the authentication proof implies a correct verification of the user's certificate. The failure to verify that the submited certificate has been signed by a trusted authority has been the main cause of the publicly known ID card authentication bypass flaws [6, 7, 8, 9, 10]. Below we describe one possible solution of how such a hardended authentication protocol could be implemented.

The basic idea of the solution is to change the protocol such that a malicious user could not submit their fake certificate to a vulnerable website that fails to correctly verify the authenticity of the certificate. Instead of returning the user's certificate, the Web eID extension would instead return only a reference to the user's certificate. This reference would then be used by the website to obtain the user's certificate from a trusted source. In practice, the certificate reference could be the name of the CA that issued the certificate and the serial number of the certificate. Using these two fields, the website can form an OCSP request to check the revocation status of the user's certificate. What then remains is the modification of the OCSP responder to return the user's certificate in the event the queried certificate is valid[3]. **As a result, in order to verify the signature of the authentication proof, the website would be forced to obtain the corresponding certificate from a trusted source, thereby also verifying the revocation status of the certificate**.

6 Concluding statements

The Web eID authentication solution is intended to replace the current TLS client certificate authentication solution in all e-services in Estonia and possibly to be used as a solution abroad as well. Since the solution will be used to protect loads of sensitive personal data and hence the well being of Estonian society, the security aspects of the solution should not be underestimated.

In some cases it may be justified to sacrifice security in favor of improved user experience. However, none of the security improvements suggested in this paper affect the user experience, hence we see no convincing reasons to discard them.

This paper serves as a warning to the designers of the Web eID solution. In the event our advice is discarded and any of the security risks listed above later materialize, the designers of the solution will have to carry moral responsibility for sacrificing security over false claims of standards-compliance.

[3]One of the options is to return the user's certificate in the unsigned `certs` field [11] of the OCSP response.

Acknowledgements. This research has been carried out with financial support from the European Social Fund through the IT Academy programme and from the Estonian Ministry of Economic Affairs and Communications.

References

[1] Mart Sõmermaa. Web eID: electronic identity cards on the Web, June 3, 2019. `https://github.com/web-eid/web-eid-system-architecture-doc`.

[2] Arnis Parsovs. ID card (browser signing extension) authentication man-in-the-middle attack, February 3, 2021. `https://www.youtube.com/watch?v=Qr638sbaZ_M`.

[3] A. Popov, M. Nystroem, D. Balfanz, and J. Hodges. The Token Binding Protocol Version 1.0. RFC 8471 (Proposed Standard), October 2018. `http://www.ietf.org/rfc/rfc8471.txt`.

[4] Cybernetica AS. Analysis of planned architectural changes in Open-eID, December 18, 2020. `https://web-eid.gitlab.io/analysis/webextensions-main.pdf`.

[5] M. Jones, J. Bradley, and N. Sakimura. JSON Web Signature (JWS). RFC 7515 (Proposed Standard), May 2015. `http://www.ietf.org/rfc/rfc7515.txt`.

[6] Arnis Parsovs. Swedbank Estonia Internet bank ID card authentication bypass, August 3, 2016. `https://www.youtube.com/watch?v=5mWMu0KinFg`.

[7] Arnis Parsovs. SEB Estonia Internet bank ID card authentication bypass, August 25, 2015. `https://www.youtube.com/watch?v=rRB8jZnS5nY`.

[8] Semjon Kravtšenko. Coop Pank Internet bank ID card authentication bypass, February 16, 2021. `https://www.youtube.com/watch?v=c0bPmkK7zaY`.

[9] Semjon Kravtšenko. elisa.ee, printincity.ee, arved.ee: ID card authentication bypass, February 18, 2021. `https://www.youtube.com/watch?v=IQ5UK2VwN4w`.

[10] ERR News. Hacker downloads close to 300,000 personal ID photos, July 28, 2021. `https://news.err.ee/1608291072/hacker-downloads-close-to-300-000-personal-id-photos`.

[11] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960 (Proposed Standard), June 2013. `https://tools.ietf.org/html/rfc6960`.